Regression Splines - Theory and Graphical Analysis

Microeconometrics and Application: Report 2

Simrit Rattan (27720199656249)
Elena Riccarda Ziege (27720199656247)

November 9, 2019

1 Statistical Background

Regression splines belong to the family of nonparametric regression models, with the aim to flexibilize modelling of the continous covariates (z) effect on the response variable (y). The problem with the pure linear model is that it cannot sufficiently explain the relation between z and y. Transformations and polynomials belong to the nonlinear regression model family, but both are insufficient for more complex relations. Here, nonparametric methods can be helpful. A way to identify nonlinear relations is to look at the scatter plot of residuals, especially the partial residual plots. We are only focusing on the effect of one covariate on the dependent variable here. The modeling approach is also known as *Scatter Plot Smoothers*. Our aim is to explain y through a deterministic function of z as following:

$$y_i = f(z_i) + \epsilon_i \tag{1}$$

By having different assumptions regarding the deterministic function we get different modeling possibilities. The problem arising now is that the estimation can be very difficult and challenging, which is why qualitative constrains, i.e. continuity and differentiability, concerning the smoothness of the deterministic function are introduced. Moreover, we have to makes some assumptions regarding the moments of the error term and the response variable.[1] (cf. Fahrmeier, et al., 2013, p. 413 ff.)

1.1 Polynomial Splines

Polynomial splines, or as they are also called regression splines, are closely related to the polynomial regression model. The standard polynomial model has the following deterministic function:

$$f(z_i) = \gamma_0 + \gamma_1 z_i + ... + \gamma_l z_i^l \tag{2}$$

Here, the relation between y and the z is modeled as a polynomial of degree l. The regression coefficients of the polynomial (γ) can be estimated by OLS, as we are still in the scope of linear models. The issue with a pure polynomial model is, that with increasing complexity it becomes insufficient to explain the relation of interest. (cf. Fahrmeier, et al., 2013, p. 413 ff.) This issue can be solved by introducing piecewise-polynomial, where the domain of z is partioned into intervals, where we separatly estimate polynomials in each interval. OLS can still be used to estimate the coefficients. Nevertheless, also the piecewise-polynomial method does not come without any issue. Due to partitioning the domain of z, there is no overall smoothness as we are having different function values at the interval boundaries, thus requiring smoothness restrictions. As soon as this is required, we start talking about *Polynomial Splines*. The underlying methodology is the same as before, however we require the function to be $(l-1)$-times continously differentiable at the interval boundaries. The intervals are referred to as knots ($a = k_1 < ... < k_m = b$) and the function is a polynomial of degree $l0$. The more knots we have, the more piecewise polynomials we estimate that together result in the polynomial spline. (cf. Fahrmeier, et al., 2013, p. 413 ff.)
Before we can use polynomial splines we need to define the representation of the set of polynomial splines for a given degree and knots configuration. There exist several different bases used to represent the space of spline functions of a certain order and knot sequence of which we will introduce the TP-Basis functions

[1] $E(\epsilon_i) = 0$ *and* $Var(\epsilon_i) = \sigma^2$ *for* $i = 1, ..., nE(y_i) = f(z_i)$ $Var(y_i) = \sigma^2$ *for* $i = 1, ..., n$

as well as B-Spline basis functions in the next two sections. Before that, let us first talk about the choice of the number and position of the knots as well as the spline degree. Later, in the empirical part we will see the importance of this when modeling the effect of z on y. There is not really a rule of thumb for the degree of splines, although the cubic spline, which produces a smooth, twice continously differantiable function, is often used as a default. Regarding the number of knots it can be said that, a large number of knots introduces greater flexibility but also greater variability and a small number of knots results in smoother estimates but also bias (Bias-Variance-Tradeoff,). The best way to decide on the number of knots is to graphically investigate different choices and see which one fits better. (cf. Hastie, et al., 2001, p. 219 ff.)

A numerical solution to this would be to use penalization or adaptive selection, which are not in the scope of this assignment. To specify the position of the knots three common approaches are introduced:

- Equidistant knots: Default option, where the domain $[a, b]$ is split into $m-1$ intervals of width $h = \frac{m-1}{b-a}$. Then the knots are $k_j = a + (j-1)h$ where $j = 1, ..., m$.
- Quantile-based knots: The $(j-1)/(m-1)$ - quantiles of the observed covariates $(z_1, ..., z_n)$ are used as knots. The advantage of this approach is that many knots are put in areas with a large number of observations, adapting better to the underlying distribution.
- Visual knot choice based on a scatter plot (cf. Fahrmeier, et al., 2013, p. 413 ff.)

1.1.1 Truncated Power Series

The TP-Basis function is a simple and easily explainable approach where the deterministic function is decomposed of a global polynomial part and a truncated polynomial part:

$$f(z_i) = \gamma_1 + \gamma_2 z_i + ... + \gamma_{l+1} z_i^l + \gamma_{l+2}(z_i - k_2)_+^l + ... + \gamma_{m+l-1}(z_i - k_{m-1})_+^l \tag{3}$$

Using this as our deterministic function fulfills the demand for global smoothnes. The truncated polynomials ensure that the change in the slope is smooth enough, so that the properties of the polynomial splines are maintained. Equation (5) can also be written as a linear combination of the $d = m + l - 1$ functions:

$$y_i = \sum_{j=1}^{d} \gamma_j B_j(z_i) + \epsilon_i \tag{4}$$

where

$$B_1(z) = 1 \qquad B_2(z) = z \qquad B_{l+1}(z) = z^l \qquad B_{l+2}(z) = (z - k_2)_+^l \qquad B_d(z) = (z - k_{m-1})_+^l \tag{5}$$

We can write all basis functions $(B_1, ...B_d)$ in one matrix, denoted as Z,

$$Z = \begin{pmatrix} B_1(z_1) & ... & B_d(z_1) \\ . & & . \\ . & & . \\ . & & . \\ B_1(z_n) & ... & B_d(z_n) \end{pmatrix} \tag{6}$$

and write equation (6) as following:

$$\boldsymbol{y} = \boldsymbol{Z\gamma} + \boldsymbol{\epsilon} \tag{7}$$

The estimated coefficients are then obtained by OLS:

$$\widehat{\gamma} = (\boldsymbol{Z'Z})^{-1}\boldsymbol{Z'y} \tag{8}$$

(cf. Fahrmeier, et al., 2013, p. 413 ff.)

1.1.2 B-Splines

The second approach is an alternative to the TP-Basis, as the TP-Basis has some numerical instabilities when z has large values. Moreover, when two knots are too close to each other the TP-Basis functions are nearly linearly dependent. The underlying methodology does not change. Piecewise polynomials are fused together at the knots in a smooth way to create the B-Splines basis functions, consisting of $(l+1)$ polynomial pieces of degree l, which are joined together in an $(l-1)$ times continously differentiable way. B-Spline basis functions are locally defined, only positive on an interval based on $(l+2)$ knots, bounded from above and $\sum_{j=1}^{d} B_j(z) = 1$, to ensure we don't have to face any numerical problems as with TP-Basis. The B-Spline basis functions are defined as following. (cf. Fahrmeier, et al., 2013, p. 413 ff.)
Degree 0:

$$B_j^0(z) = \mathbf{1}[k_j, k_{j+1}](z), \qquad j = 1, ..., d = m - 1 \tag{9}$$

$$= \begin{cases} 1, & k_j z < k_{j+1} \\ 0, & otherwise \end{cases} \tag{10}$$

For each increase in degree l, 2 external knots k_{-l+1}, k_{m+1} are added, giving us higher order B-Splines:

$$B_j^l(z) = \frac{z - k_{j-l}}{k_j - k_{j-l}} B_{j-1}^{l-1}(z) + \frac{k_{j+1} - z}{k_{j+1} - k_{j+1-l}} B_j^{l-1}(z) \tag{11}$$

Compared to the TP-Basis functions we also have $2l$ knots outside of the domain $[a, b]$. This approach might seem to be more complicated than the TP-Basis, but the advantage of the B-Spline basis functions, are the already mentioned better numerical properties. Equation (8), (9) and (10) are equivalent for the B-Spline basis, with just the basis functions being different in the $\boldsymbol{Z}$ matrix. (cf. Fahrmeier, et al., 2013, p. 413 ff.)

2 Empirical Implementation

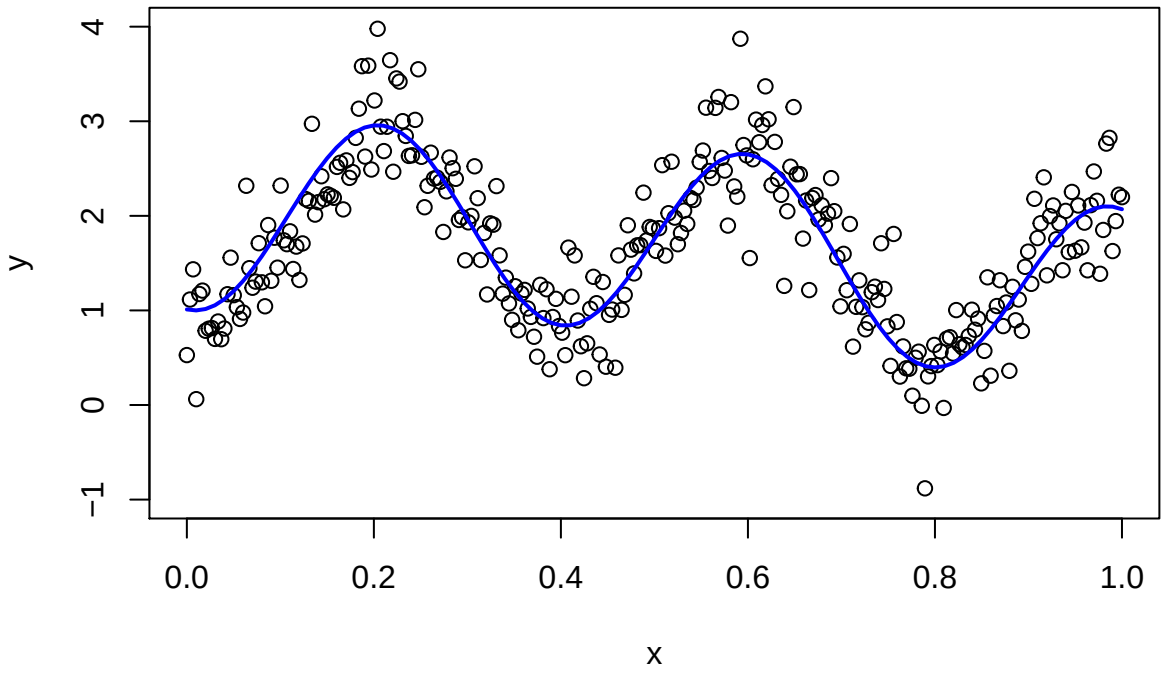
2.1 Simulation

We chose to use 300 observations to simulate the training data using normal distributed errors with variance 0.4^2. We define the function

$$f(z) = \sin(4(4x - 2)) + 2\cos(x) \tag{12}$$

as our true function. We used x as our covariate instead of z which we used in the explanations above. The blue curve in the graph depicts our true funtion.

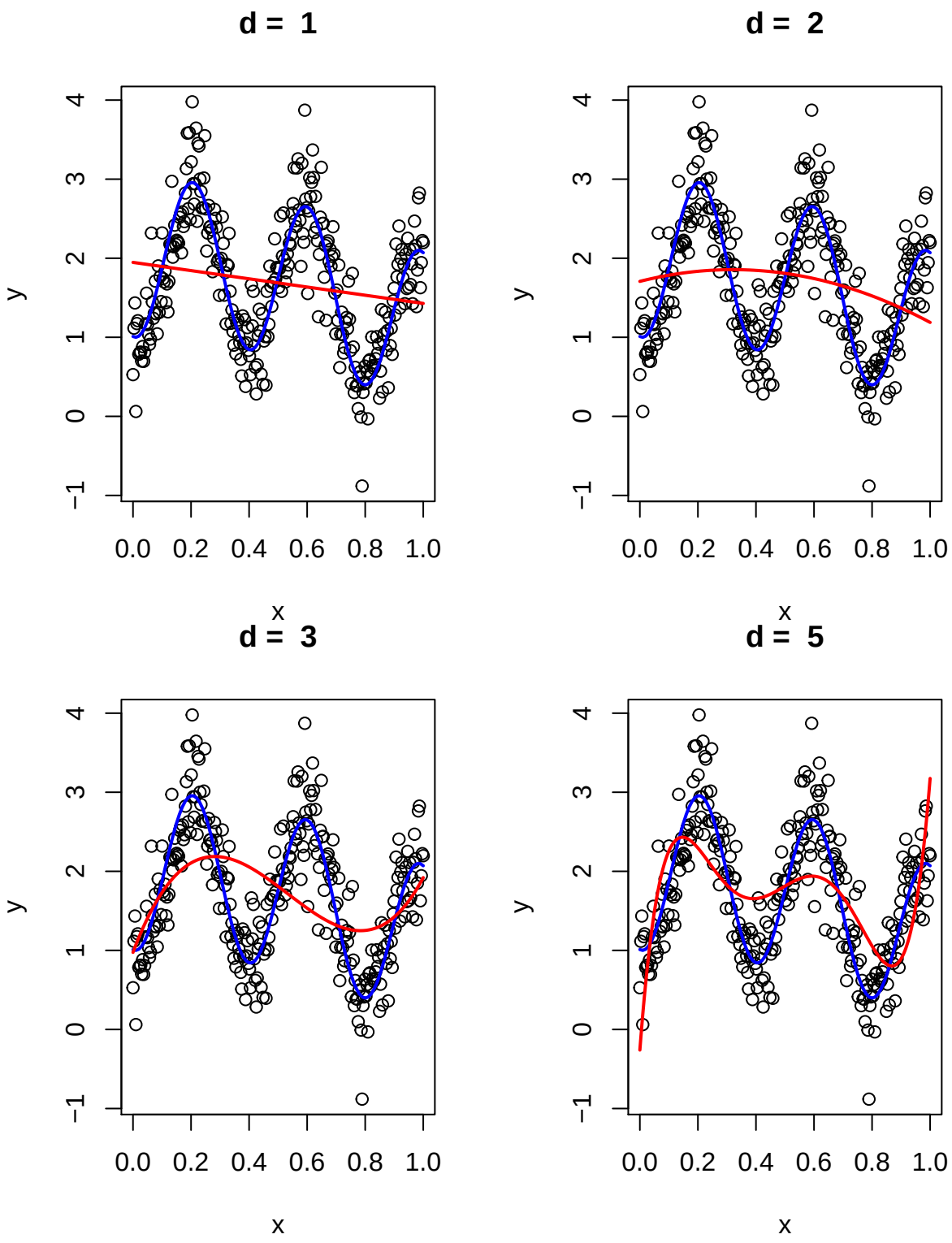
```
set.seed(1234)

n <- 300
f <- function(x){
sin(4*(4*x - 2)) + 2*cos(x) }
x <- seq(from = 0, to = 1, length.out = n)
epsilon <- rnorm(n, 0, 0.4)
y <- f(x) + epsilon
plot(x, y, ylim = c(-1, 4))
curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
```

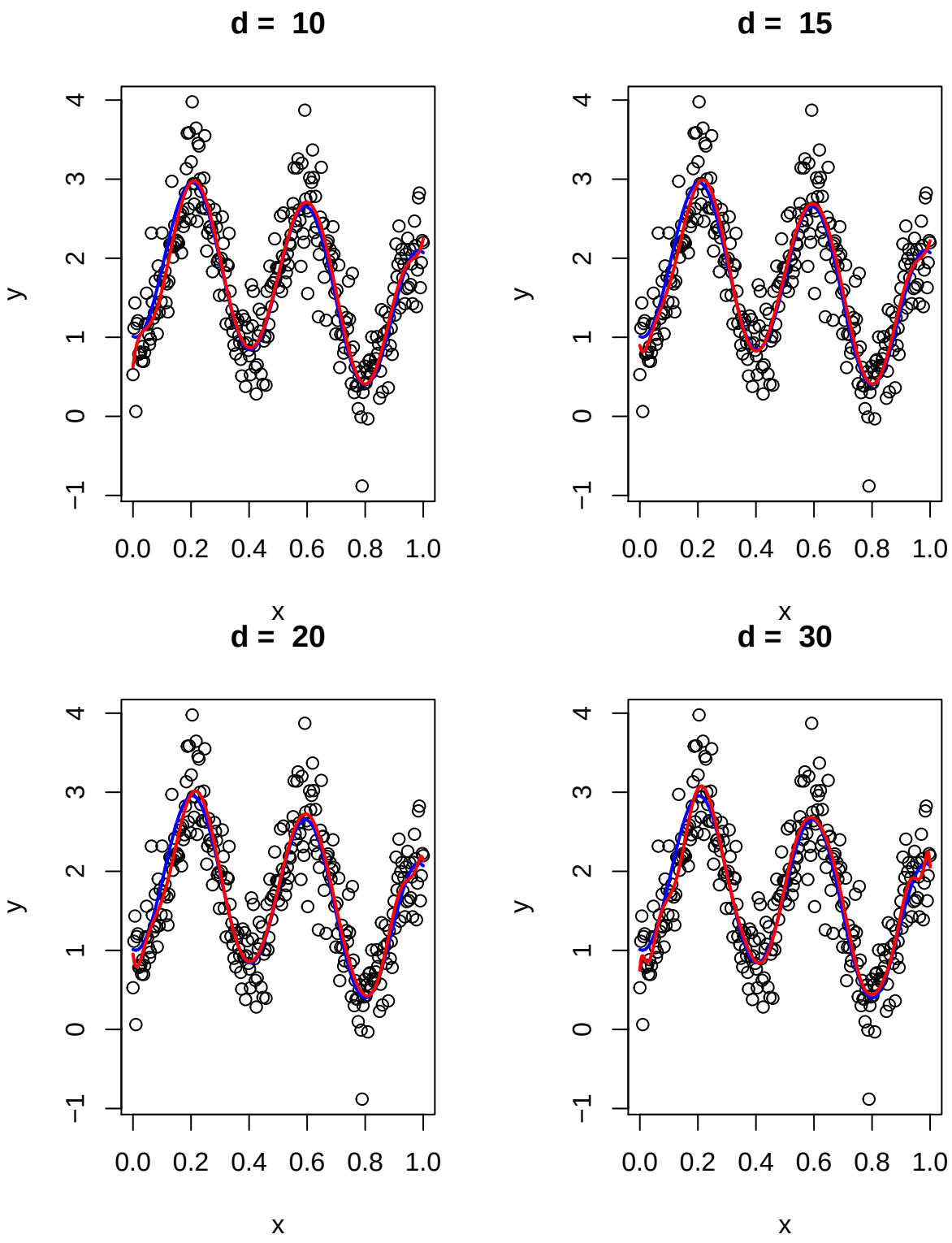


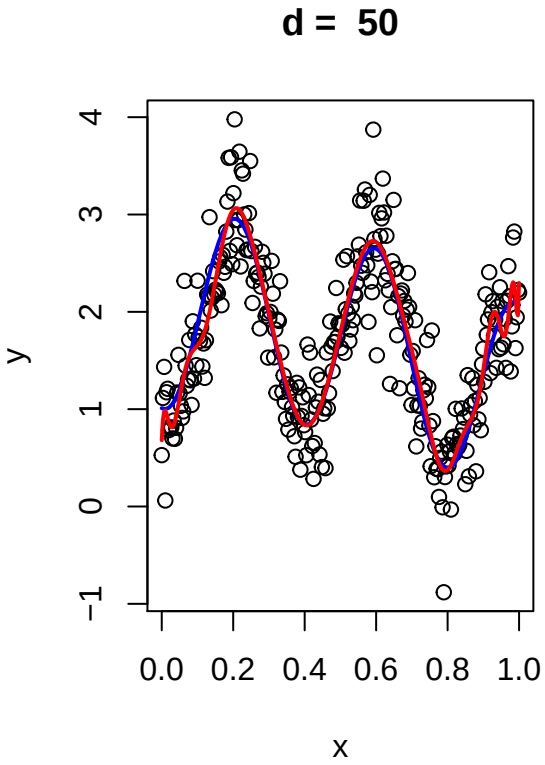
2.2 Polynomial Model

First we define the function `polynom_Fit()` which we later use on our training data set *train* to fit several polynomial models with different degrees.

```
polynom_Fit <- function(d, y = y, x = x){
train <- data.frame(y = y)
for(i in 1:d) {
    train <- cbind(train, x^i)
names(train)[length(train)] <- paste0("x^",i) }
  fit <- lm(y ~ . , data = train)
return(fit)}

degree <- c(1,2,3,5,10,15,20,30,50)
fit_list <- lapply(degree,polynom_Fit, y = y, x = x)

par(mfrow = c(1,2))
for(i in 1:9) {
  plot(x,y, main = paste("d = ", degree[i]))
  curve(f(x), 0, 1, add = TRUE, lwd =2, col="blue")
  lines(x, predict(fit_list[[i]]), lwd = 2, col = "red")
}
```


d = 1
d = 2
d = 3
d = 5
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


d = 10
d = 15
d = 20
d = 30
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0



It can be observed in the different graphs that the fitted model, which is portrayed by a red curve gets closer to the true function, portrayed in blue, with increasing number of degrees. However, once the model gets very complex (which means a high degree), the fitted curve tends to fluctuate a lot. For example for 30 degrees the fitted curve deviates from the true function for extreme values of x while the fitted model of 20 degrees fits the function very well. This tendency can be clearly seen for 50 degrees, as the fitted curve deviates too much for extreme values of x. Thus, concluding from the graphs, the polynomial model of degree 20 explains the data most accurately of the used models.

Evaluating the Polynomial Model

To evaluate the different polynomial models according to their residual sum of squares, we simulate a test data set of 300 observations with the same error and distribution as the training data set. As would be expected, the mean squared error of the function fit on the test data set decreases as complexity, here quantified in degrees of polynomial, increases, indicating that the best fit according to the MSE would be a polynomial of degree 50. But as seen in the graphical analysis above, it does not fit the data very well.

```
# Test data set
ntest <- 300
xtest <- seq(from = 0, to = 1, length.out = ntest)
epsilon_test <- rnorm(ntest, 0, 0.4)
ytest <- f(xtest) + epsilon_test
prediction_Fit <- function(d, y = ytest, x = xtest){
test <- data.frame(y = ytest)
for(i in 1:d) {
    test <- cbind(test, xtest^i)
names(test)[length(test)] <- paste0("x^",i) }
  prediction <- predict(polynom_Fit(d,ytest,xtest), newdata = test)
return(prediction)}
pred_list <- lapply(degree,prediction_Fit, y = ytest, x = xtest)
```

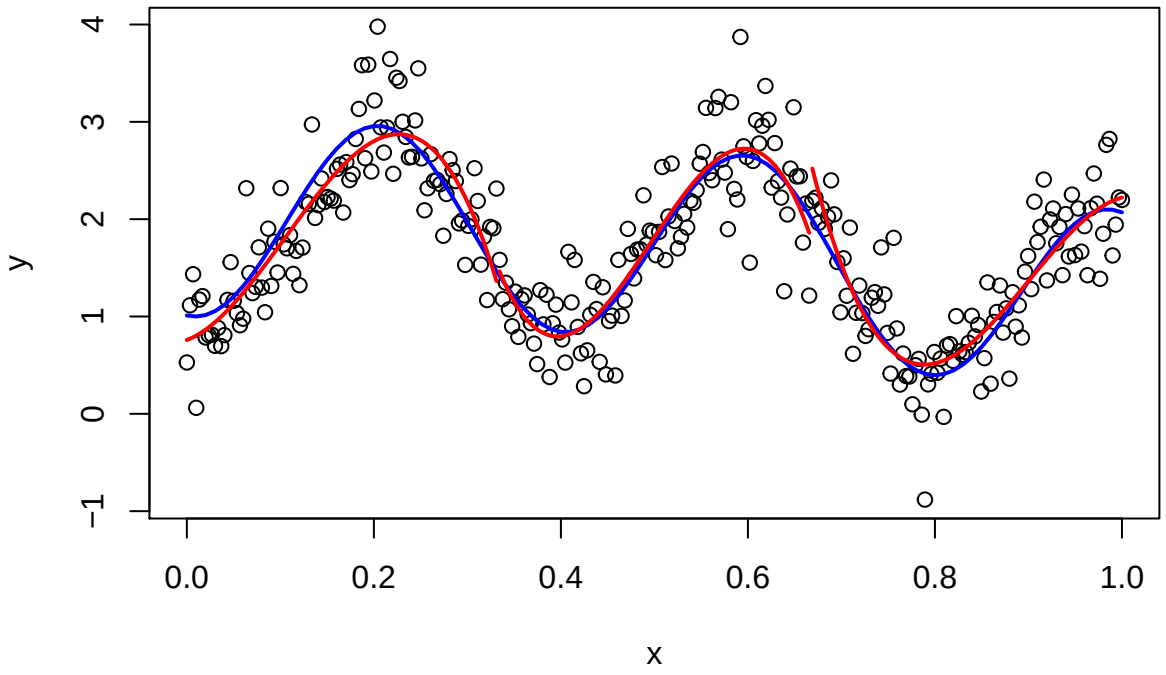
```r
# Mean Squared Error
MSE <- list()
for (i in 1:length(pred_list)) {
currentPrediction <- pred_list[[i]]
MSE[i]<- mean((currentPrediction - ytest)^2)
}

output <- cbind(MSE)
rownames(output) <- c("d=1", "d=2", "d=3", "d=5", "d=10", "d=15", "d=20", "d=30", "d=50")
output
```

```
##       MSE
## d=1  0.6989094
## d=2  0.6929929
## d=3  0.6278282
## d=5  0.4275736
## d=10 0.1665892
## d=15 0.1654421
## d=20 0.1645495
## d=30 0.1628437
## d=50 0.1606594
```

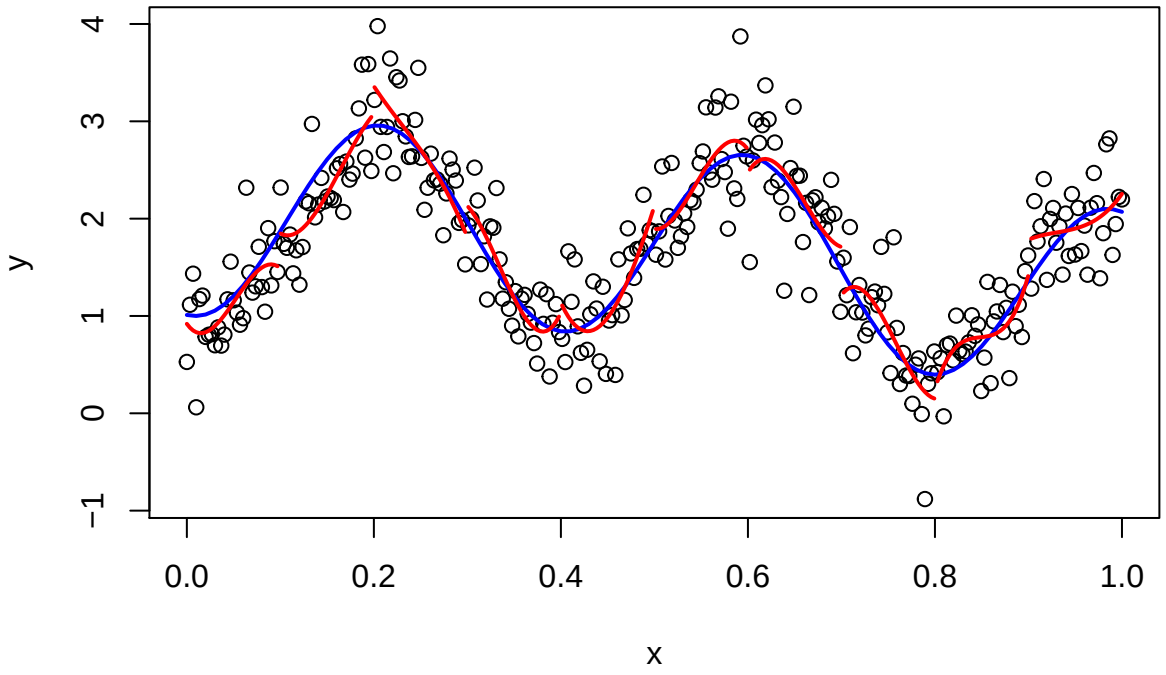
2.3 Piecewise Polynomial Model

Now we build a piecewise polynomial model, for which we divide the domain of x into different subintervals. One can see the different plotted polynmials in each subinterval. As we have discussed above, this approach does not lead to overall smoothness. There are some gaps at the boundaries of the different subintervals. We can see that a small number of intervals fits the true function better than a high number of intervals, as our true function itself is not very wiggly. The more intervals we have, the more wiggly each polynomial gets. As these gaps between the subintervals are not wanted we next continue to fit some regression splines.

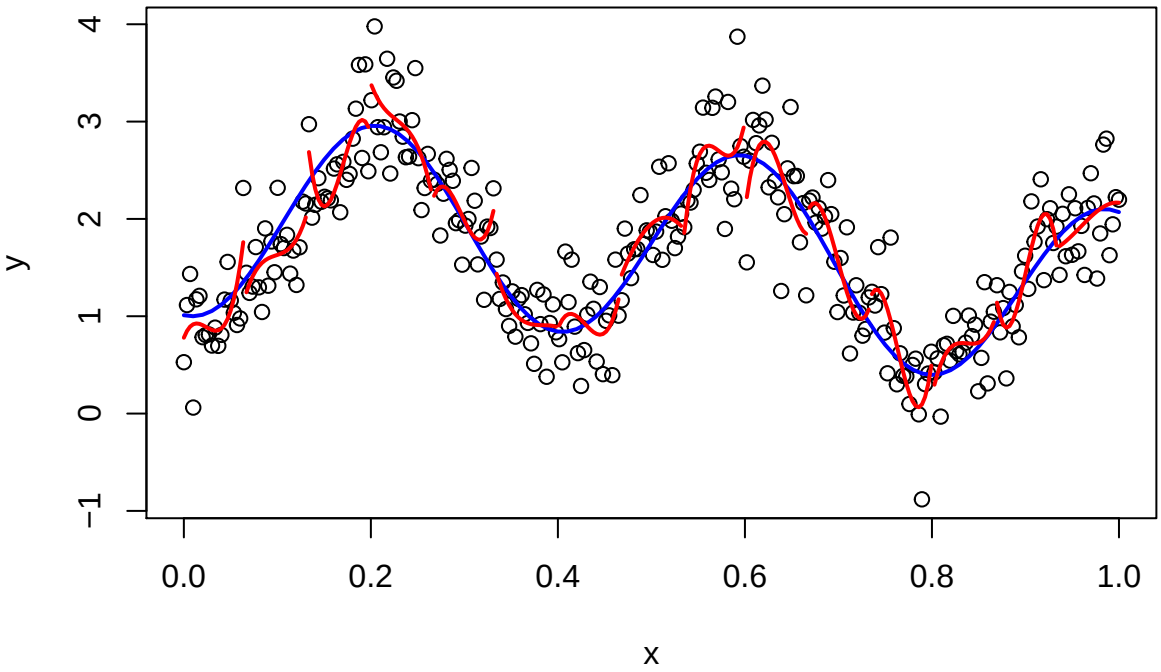
```r
# 3 intervals
sub <- matrix(1:n, nrow = 3, ncol = n / 3, byrow = TRUE)
plot(x, y)
curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
for(i in 1:3){
     lines(
       x[sub[i, ]],
       lm(y[sub[i, ]] ~ poly(x[sub[i, ]], degree = 3, raw = TRUE))$fitted.values,
       lwd = 2, col = "red")
}
```




```r
# 10 intervals
sub <- matrix(1:n, nrow = 10, ncol = n / 10, byrow = TRUE)
plot(x, y)
curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
for(i in 1:10){
     lines(
       x[sub[i, ]],
       lm(y[sub[i, ]] ~ poly(x[sub[i, ]], degree = 3, raw = TRUE))$fitted.values,
       lwd = 2, col = "red")
}
```



```r
# 15 intervals
sub <- matrix(1:n, nrow = 15, ncol = n / 15, byrow = TRUE)
plot(x, y)
curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
for(i in 1:15){
     lines(
       x[sub[i, ]],
       lm(y[sub[i, ]] ~ poly(x[sub[i, ]], degree = 3, raw = TRUE))$fitted.values,
       lwd = 2, col = "red")
}
```



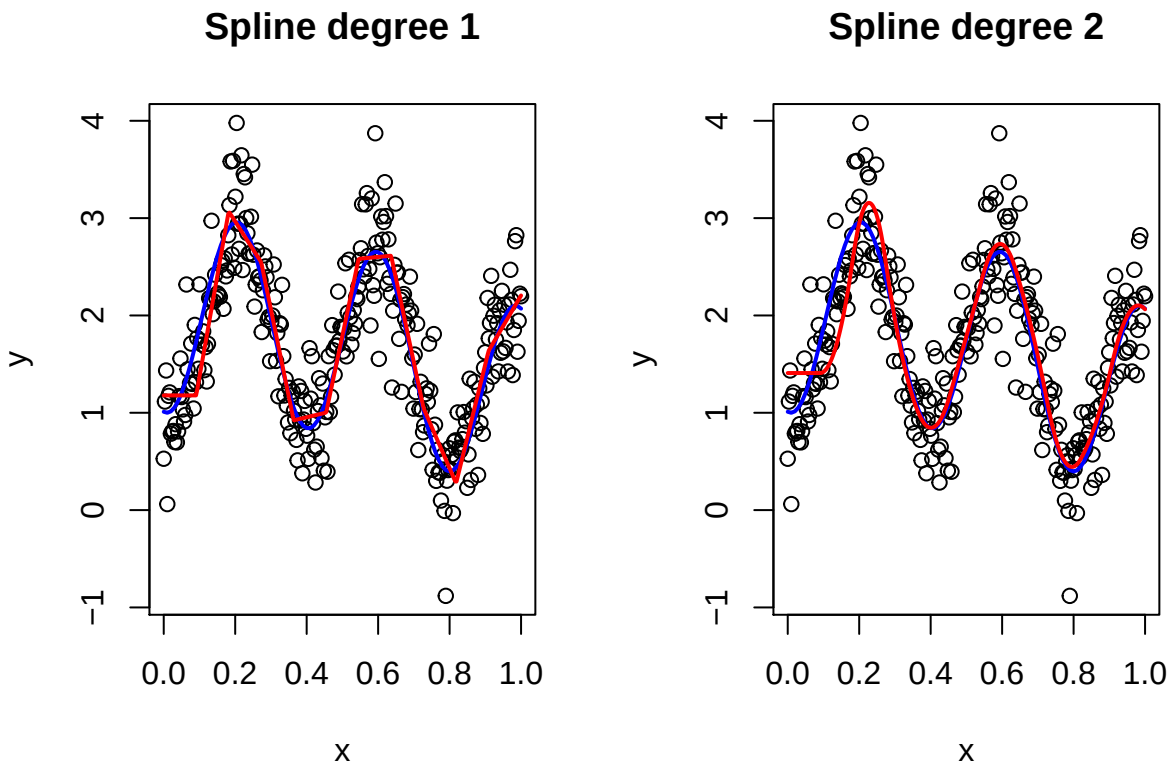
2.4 Regression Splines

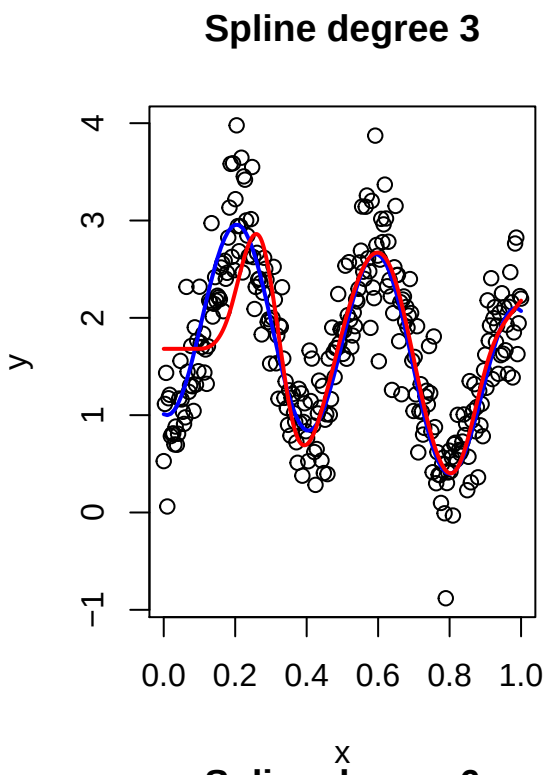
2.4.1 Trunctated Power Series

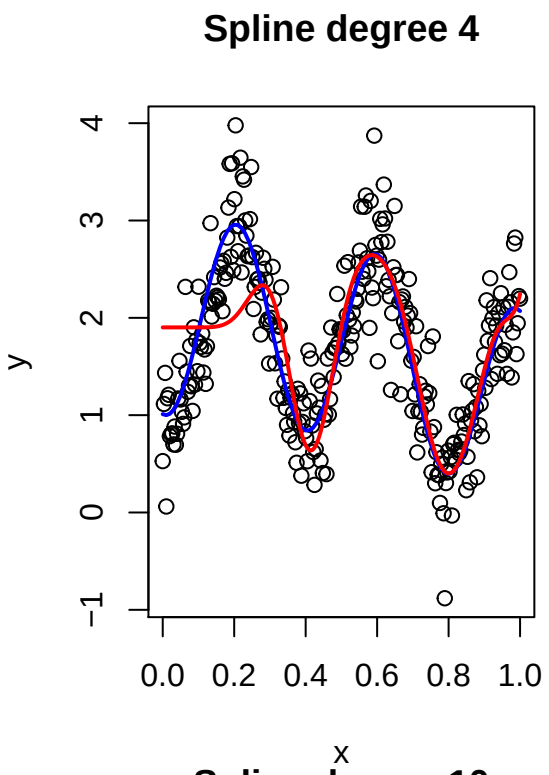
The first basis funtion that we use to construct regressin splines is the TP-Basis function as defined in section 1.1.1. We use the function `tp()` to determine the basis. Here, k is the sum of the number of knots and the number of degrees. Firstly, we randomly set the number of knots to be 10 and plot TP splines with different numbers of degrees. Graphically, we can conclude that the TP spline with two degrees fits our true function the best. With higher degree, the TP spline fits the true function less accurately. Therefore, we next use the fixed number of two degrees two compare different numbers of knots. Now, when the number of degrees plus the number of knots equals 17 (= 15 knots), our fit is the closest to the true funtion. With higher number of knots, especially 50, the fitted curve wiggles a lot, which verifies our above stated theory.

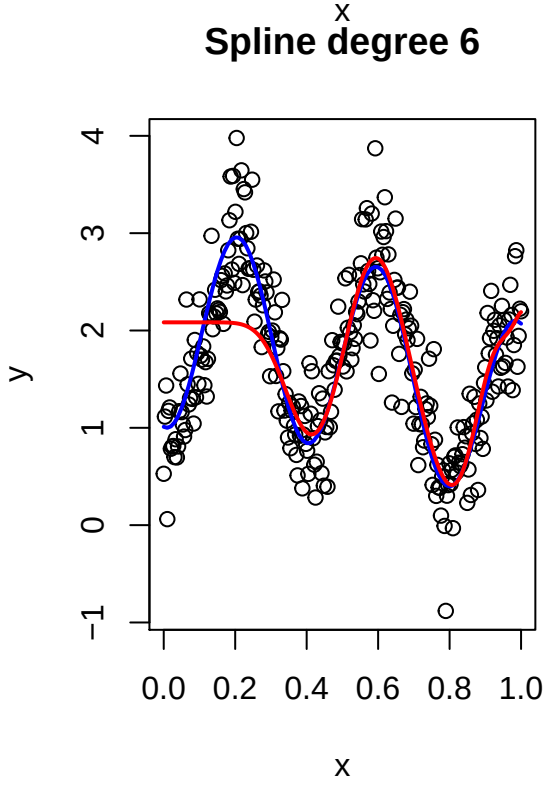
```r
library(cplm)
plot_TPspline <- function(degree, knotsanddegree, header = c("deg", "kno")){
fit <- lm(y ~ tp(x, degree = degree, k = knotsanddegree)$Z)

switch(header,
"deg" = {main <- paste("Spline degree", degree)},
"kno" = {main <- paste("# knots + degrees", knotsanddegree)})
  plot(x, y, main = main)
```

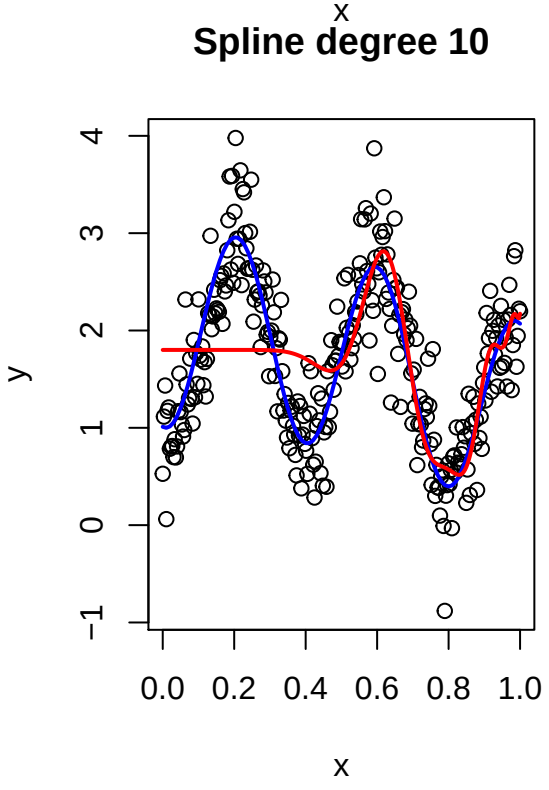
```r
  curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
  lines(x, predict(fit), lwd = 2, col = "red")

}

# Knots = 10, comparing different degrees
par(mfrow=c(1,2))
for(i in c(1, 2, 3, 4, 6, 10))
  {
plot_TPspline(degree = i, knotsanddegree = i + 10, header = "deg")
}
```



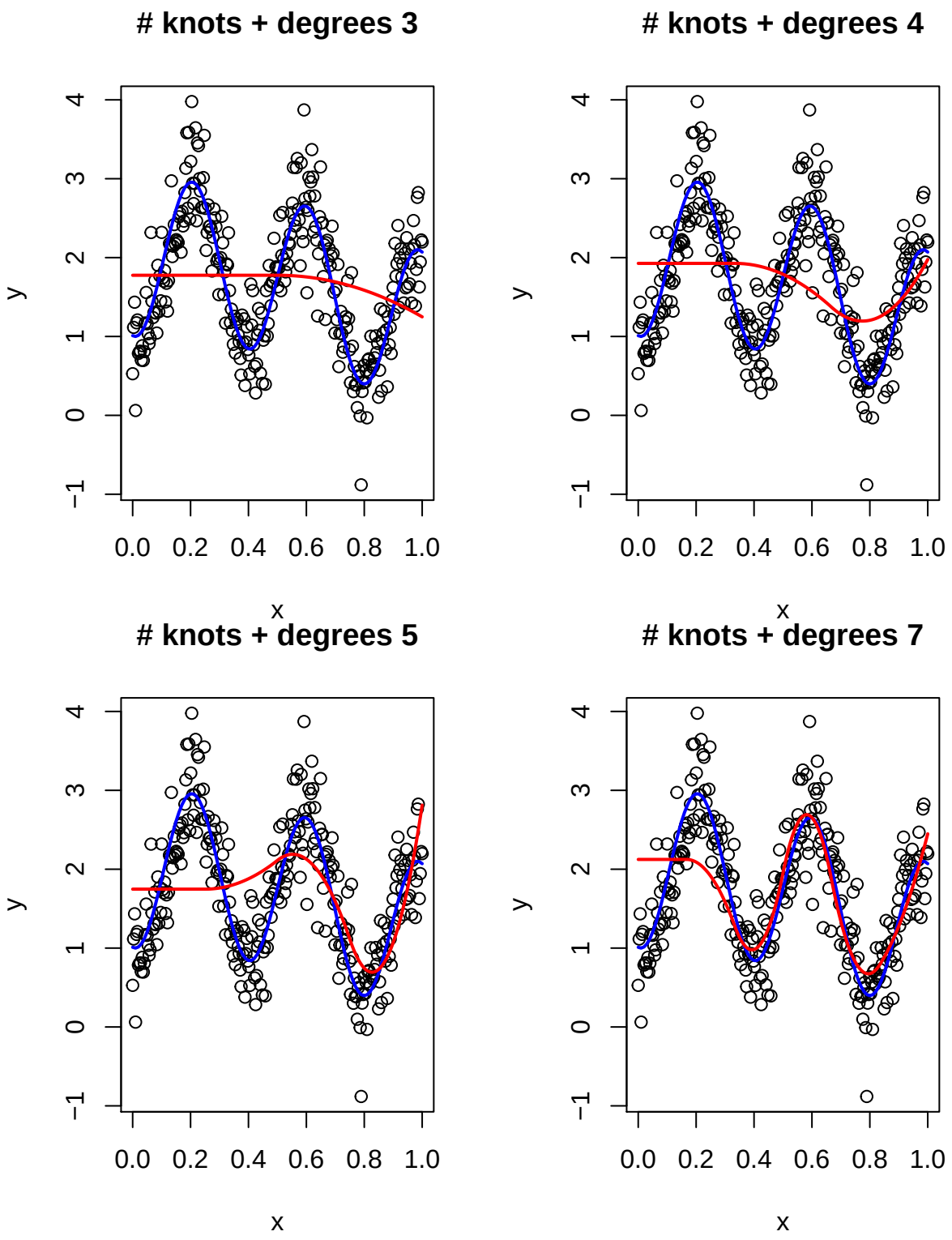


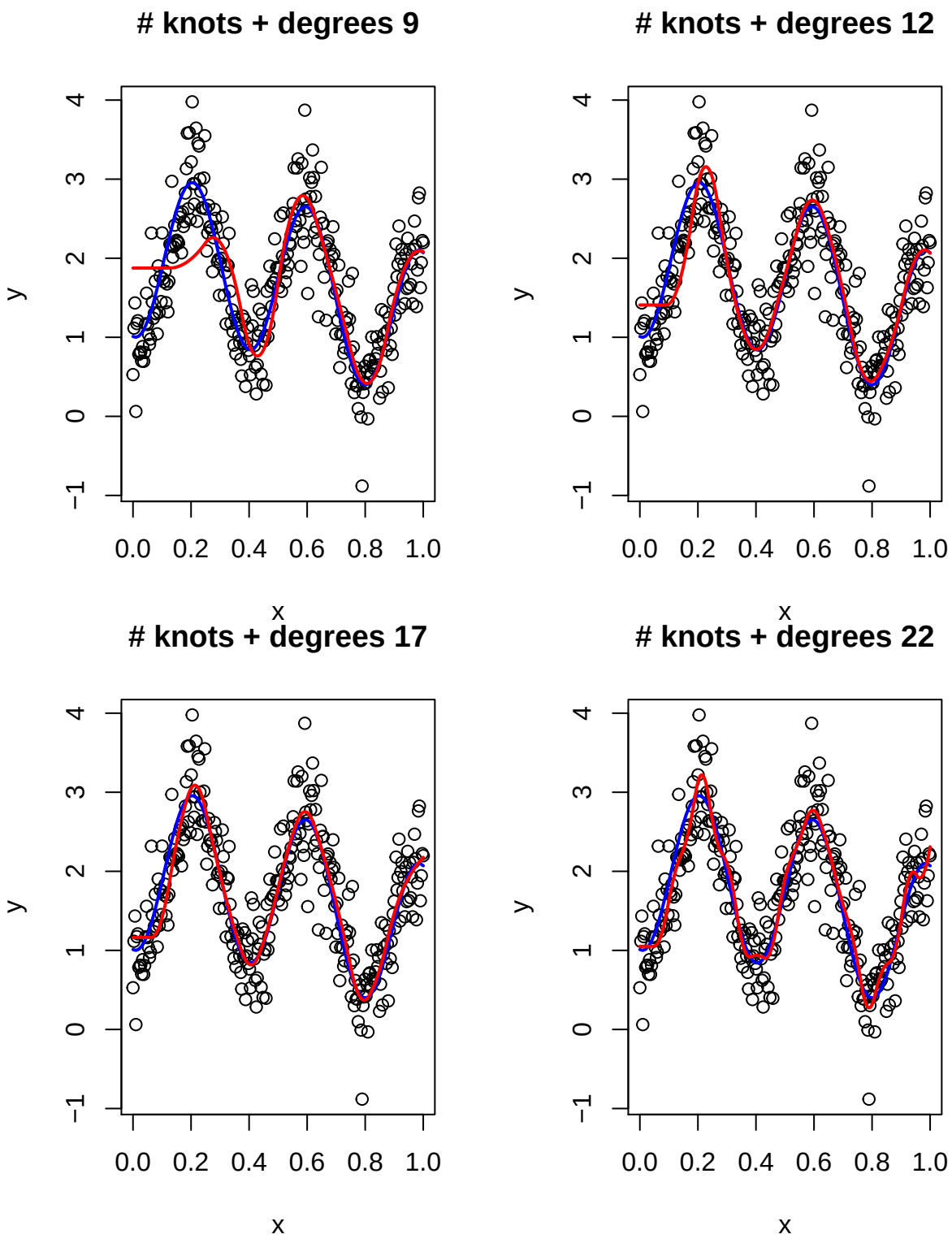


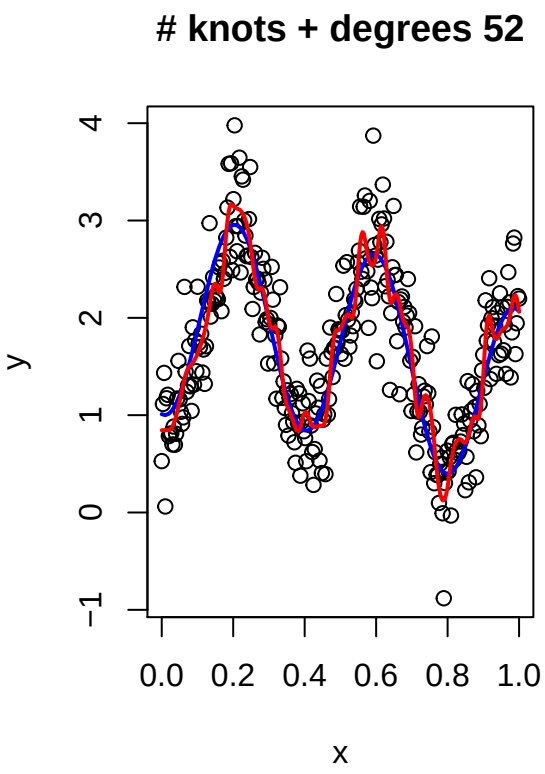





```
# Degree = 2, comparing different knots
par(mfrow=c(1,2))
for(i in c(1, 2, 3, 5, 7, 10, 15, 20, 50)){
plot_TPspline(degree = 2, knotsanddegree = i + 2, header = "kno") }
```


knots + degrees 3
knots + degrees 4
knots + degrees 5
knots + degrees 7
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


knots + degrees 9
knots + degrees 12
knots + degrees 17
knots + degrees 22
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

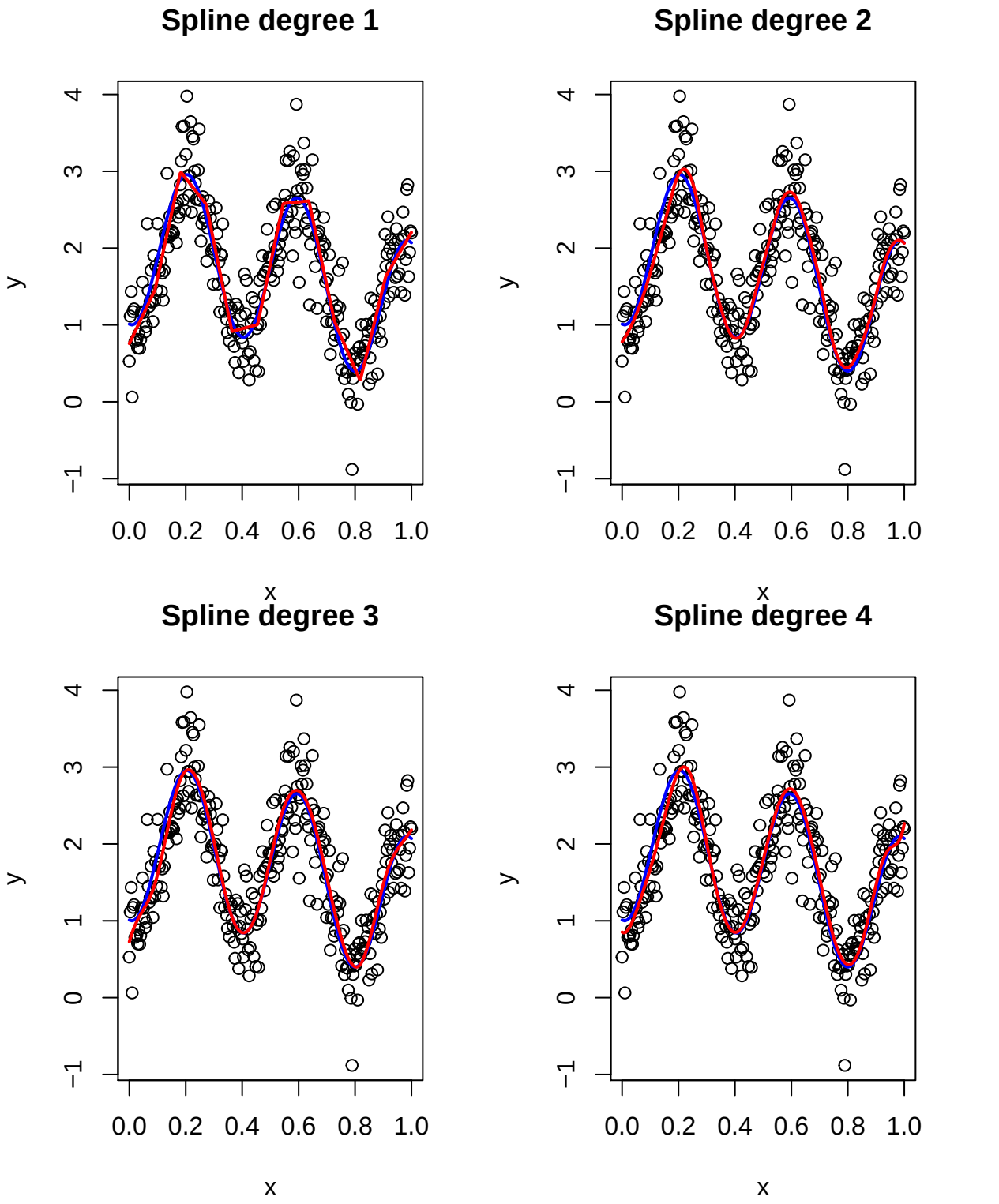


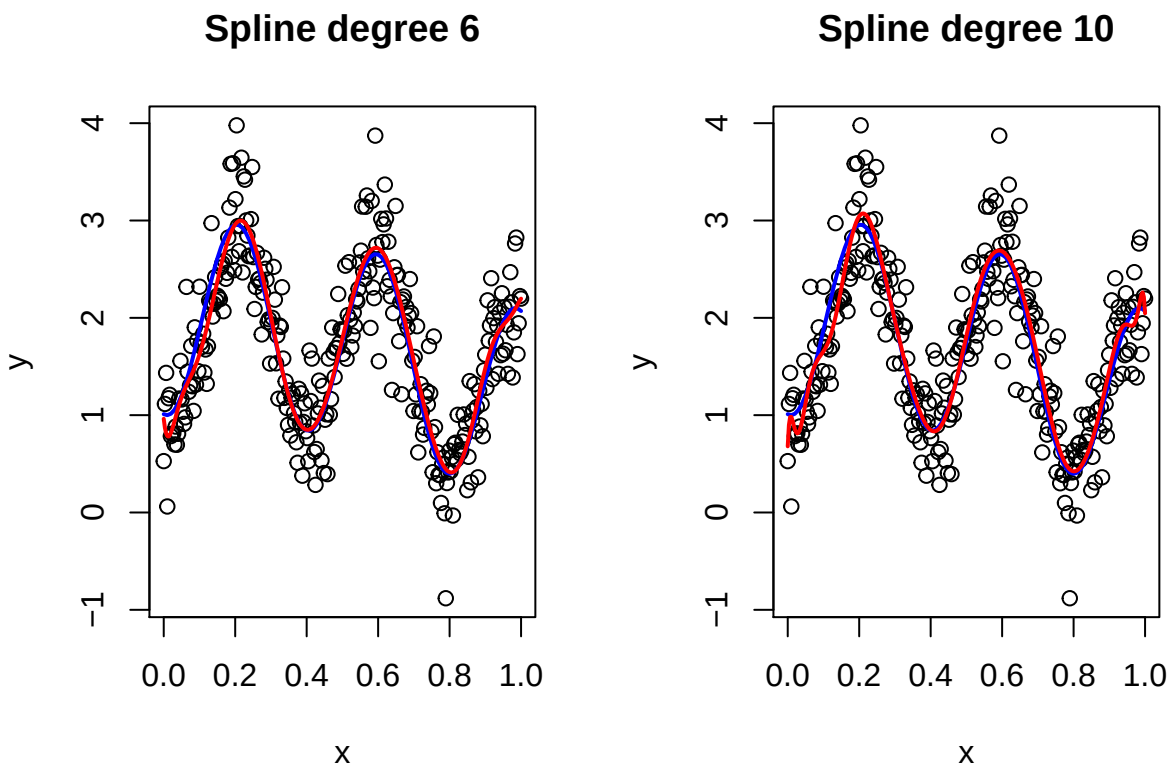
2.4.2 B-Splines

Finally, we fit different B-Splines to the training data set using the built-in command `bs()`. When comparing the different graphs, one can see that the spline of degree 3 fits our true function most accurately. Splines of lower degree have larger deviations while splines of higher degrees tend to be wiggly at the extreme values of x. This can especially be seen for degree 10 and would be expected for any degrees higher than 10. The truncated power splines on the other hand do not fit the true function well for degrees higher than 2, especially for lower values of x. Thus, the B-Splines perform better than the truncated power splines on this data set. Different than for the TP-Splines, the B-splines using 10 knots with a degree of 3 describes the data best. Similar to the TP-Splines, the models are more wiggly with higher number of knots. At first, the models only get wiggly for extreme values of x but with 50 knots the whole curve is wiggly.

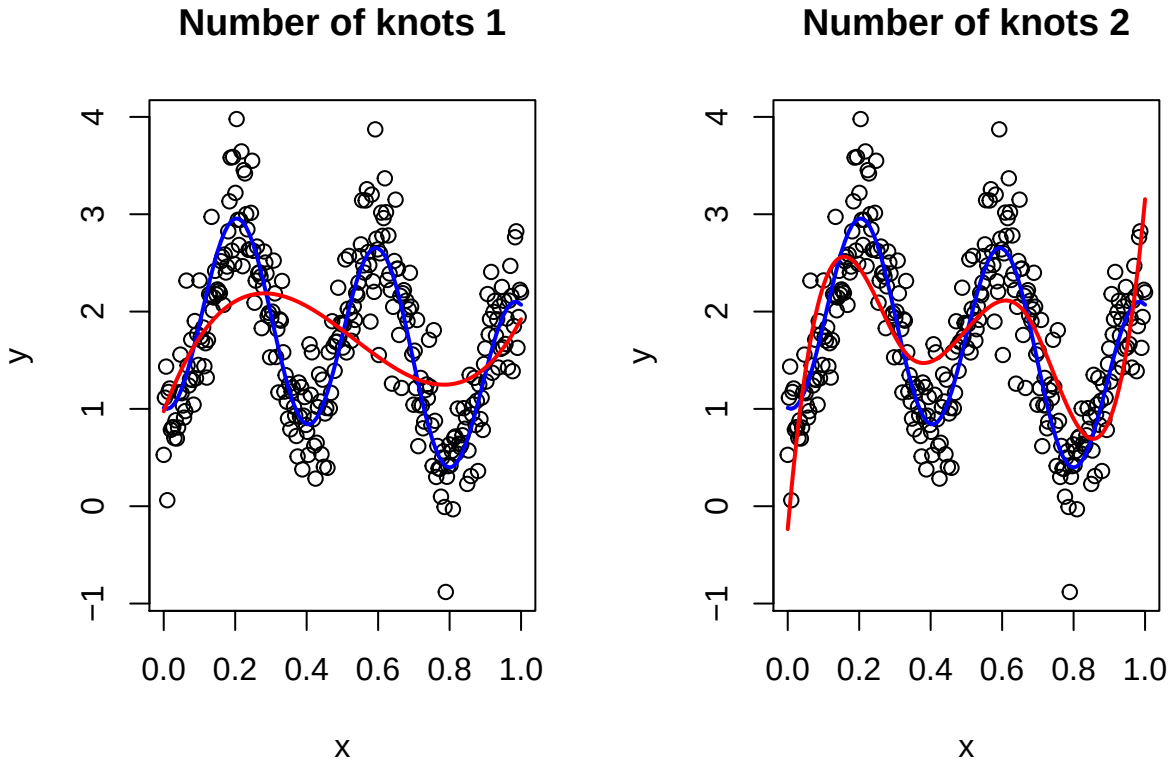
```
library(splines)
plot_BSspline <- function(degree, knots, header = c("deg", "kno")){
# estimate the b spline fit
fit <- lm(y ~ bs(x, degree = degree, df = knots + degree))

#plot the estimate
switch(header,
"deg" = {main <- paste("Spline degree", degree)},
"kno" = {main <- paste("Number of knots", knots)})
  plot(x, y, main = main)
  curve(f(x), 0, 1, add = TRUE, lwd = 2, col="blue")
  lines(x, predict(fit), lwd = 2, col = "red")

}

# Knots = 10, comparing different degrees
par(mfrow=c(1,2))
```

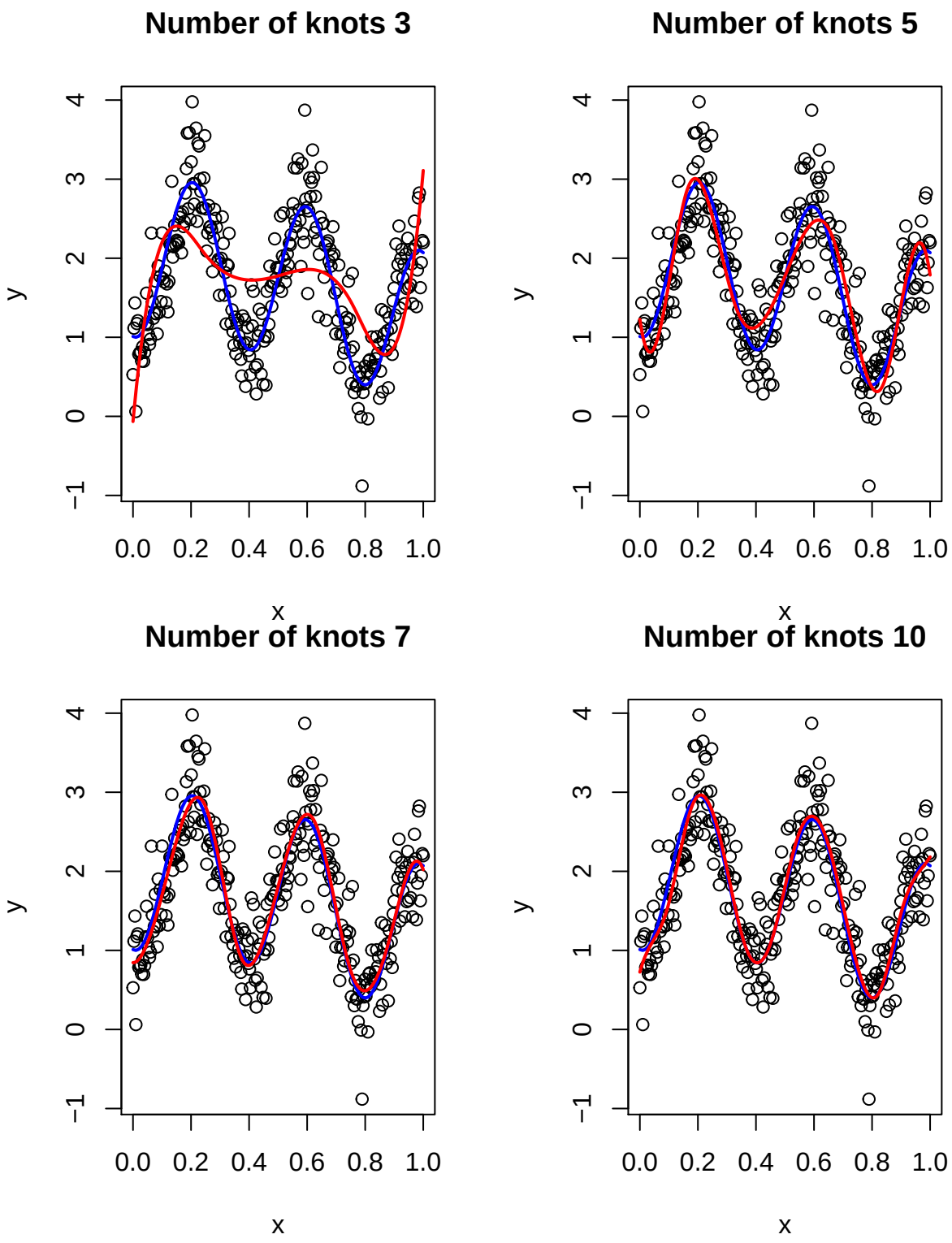
```
for(i in c(1, 2, 3, 4, 6, 10))
  {
plot_BSspline(degree = i, knots = 10, header = "deg")
}
```

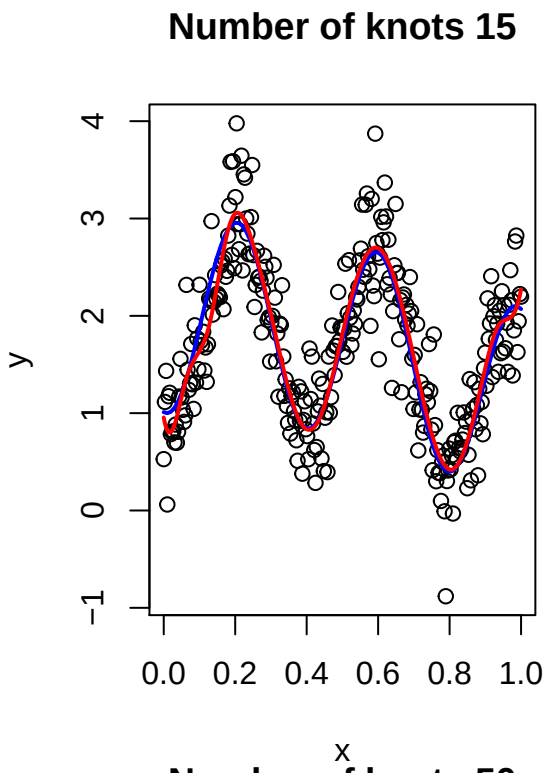


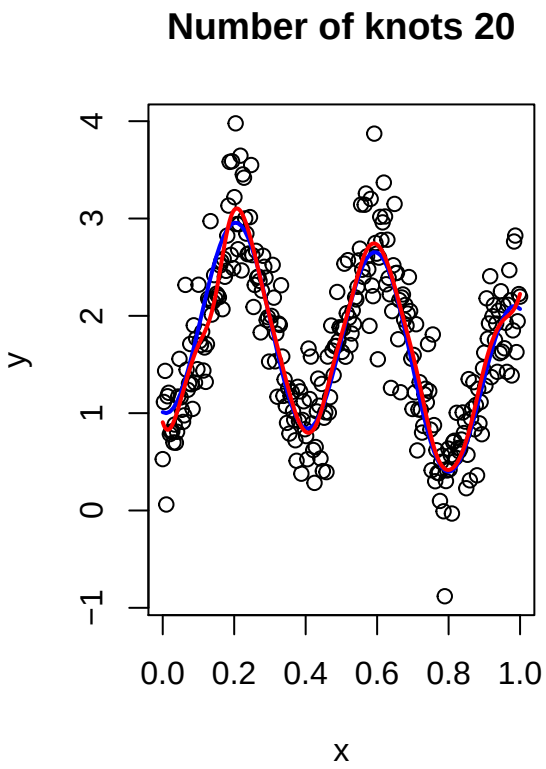


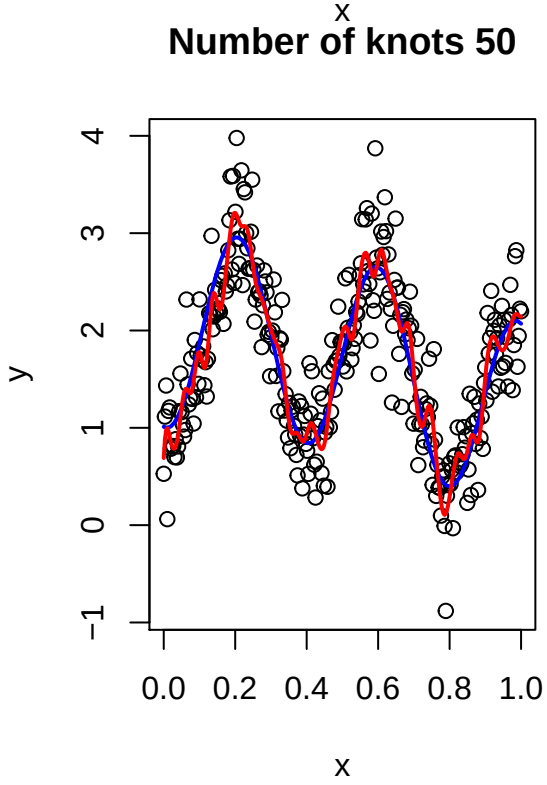

```
# Degree = 3, comparing different knots
par(mfrow=c(1,2))
for(i in c(1, 2, 3, 5, 7, 10, 15, 20, 50)){
plot_BSspline(degree = 3, knots = i, header = "kno") }
```




Number of knots 3
Number of knots 5
Number of knots 7
Number of knots 10
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


Number of knots 15
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


Number of knots 20
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


Number of knots 50
y
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

3 Literature

Fahrmeier, L., Kneib, T., Lang, S. & Marx, B., 2013. Regression - Models, Methods and Applications. New York: Springer.
Greven, S. , 2019. Lecture: Generalized Regression, Humboldt-Universität zu Berlin.
Hastie, T., Tibshirani, R. & Friedman, J., 2001. The Elements of Statistical Learning - Data Mining, Inference and Prediction. 2 Hrsg. Stanford: Springer.