

1 Semi- und Nonparametrische Regression (I)

Aufgabe 1

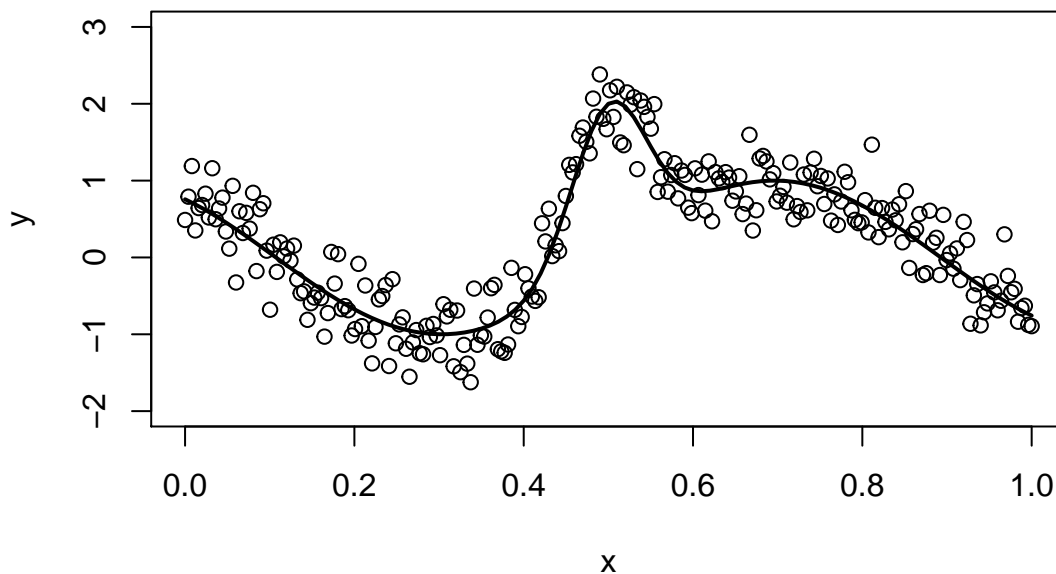
Trainings- und Testdaten erzeugen:

```
set.seed(2)

n <- 250
ntest <- 500
f <- function(x){sin(2*(4*x - 2)) + 2*exp(-(16)^2*(x - 0.5)^2)}

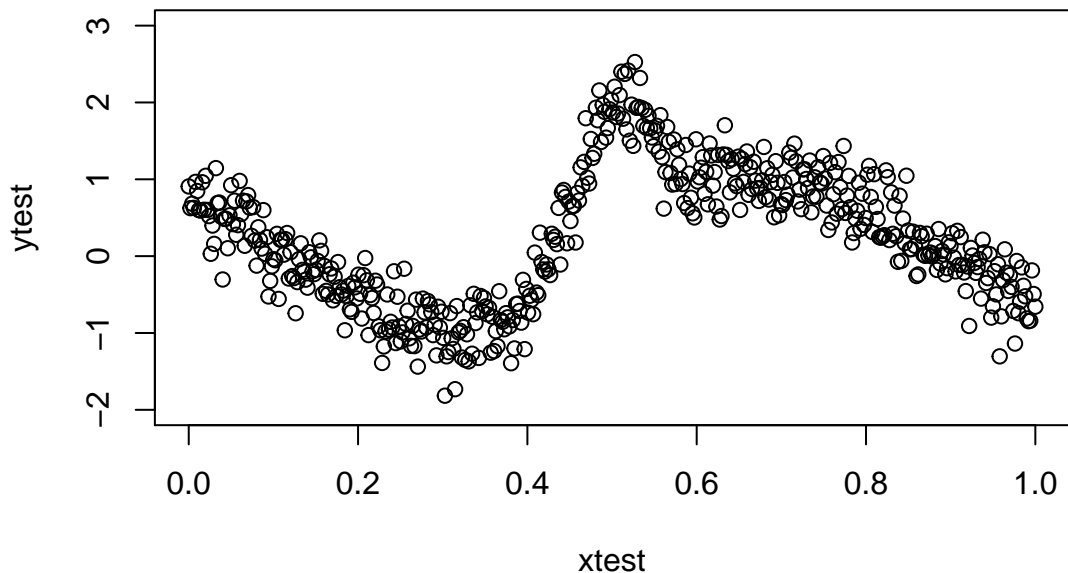
x <- seq(from=0, to=1, length.out = n)
eps <- rnorm(n, 0, 0.3)
y <- f(x) + eps

plot(x,y, ylim = c(-2,3))
curve(f(x), 0, 1, add=T, lwd = 2)
```



```
xtest <- seq(from=0, to=1, length.out = ntest)
epstest <- rnorm(ntest, 0, 0.3)
ytest <- f(xtest) + epstest

plot(xtest, ytest, ylim = c(-2,3))
```



a) Funktion zum Berechnen eines polynomialen Fits:

```

polynomfit <- function(d){
  traindata <- data.frame(y = y)
  # konstruiere die Designmatrix für den gewählten Polynomgrad
  for(i in 1:d){
    traindata <- cbind(traindata, x^i)
    names(traindata)[length(traindata)] <- paste("x^",i)
  }
  testdata <- data.frame(y = ytest)
  for(i in 1:d){
    testdata <- cbind(testdata, xtest^i)
    names(testdata)[length(testdata)] <- paste("x^",i)
  }

  fit <- lm(y ~ . , data = traindata)
  pred <- predict(fit, newdata = testdata)

  out <- list(fit = fit,
             prediction = pred,
             rss = sum(resid(fit)^2),
             rssstest = sum((ytest - pred)^2)
             )
  return(out)
}

```

Fitte ein lineares, quadratisches und kubisches Modell sowie höherdimensionale Polynome: $d = c(1, 2, 3, 5, 10, 20, 50, 100, 250, 500, 1000)$

```

# linear:
fit1 <- polynomfit(d = 1)
# quadratisch:
fit2 <- polynomfit(d = 2)
# kubisch:
fit3 <- polynomfit(d = 3)
# usw.:
fit5 <- polynomfit(d = 5)
fit10 <- polynomfit(d = 10)
fit20 <- polynomfit(d = 20)
fit50 <- polynomfit(d = 50)
fit100 <- polynomfit(d = 100)
fit250 <- polynomfit(d = 250)

```

```
fit500 <- polynomfit(d = 500)
fit1000 <- polynomfit(d = 1000)

# Plot der geschätzten Kurven:
par(mfrow=c(3,3))

plot(x, y, main = "d = 1")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit1$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 2")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit2$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 3")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit3$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 5")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit5$fit), lwd = 2, col = "red")

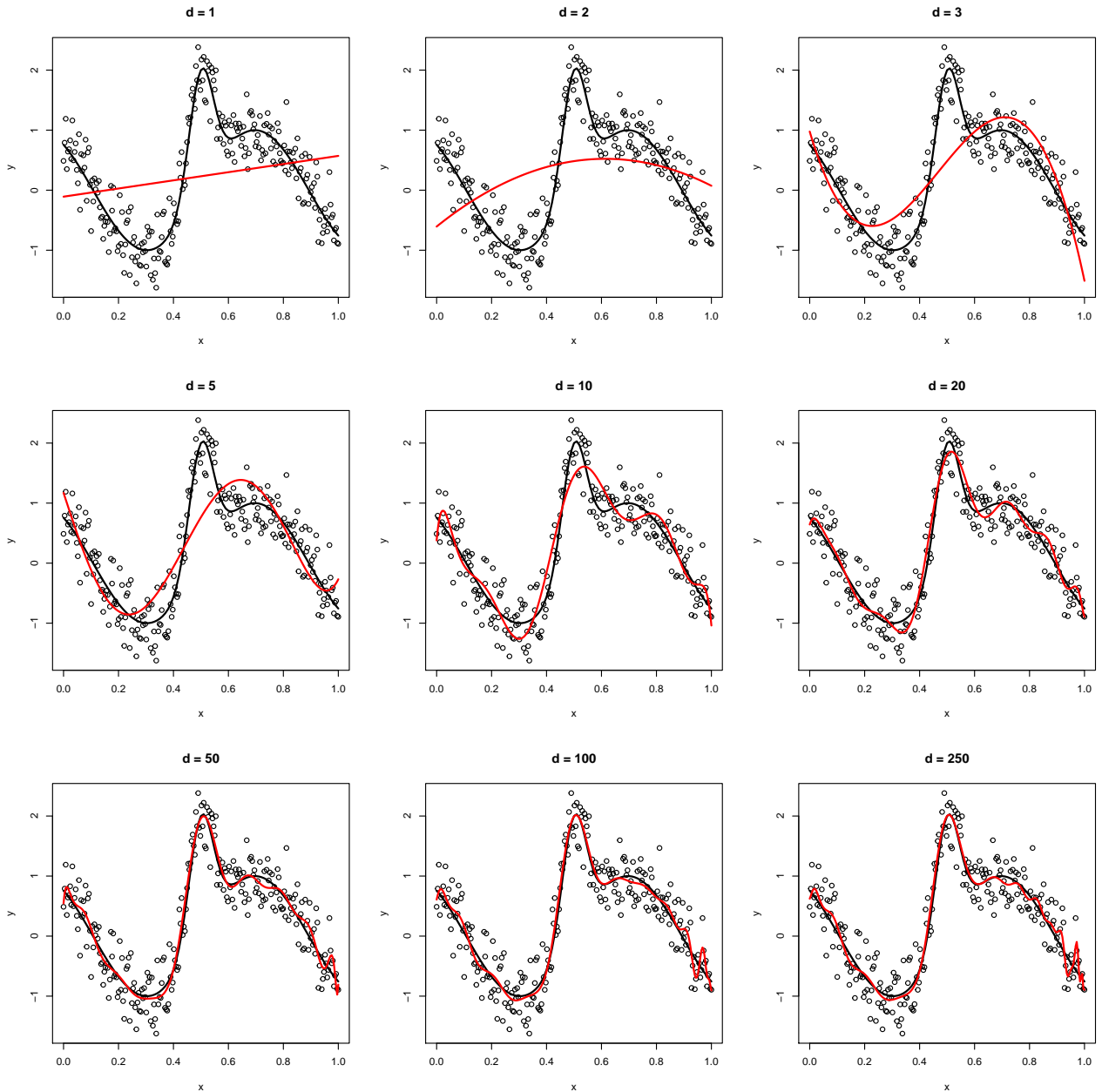
plot(x, y, main = "d = 10")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit10$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 20")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit20$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 50")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit50$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 100")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit100$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 250")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit250$fit), lwd = 2, col = "red")
```



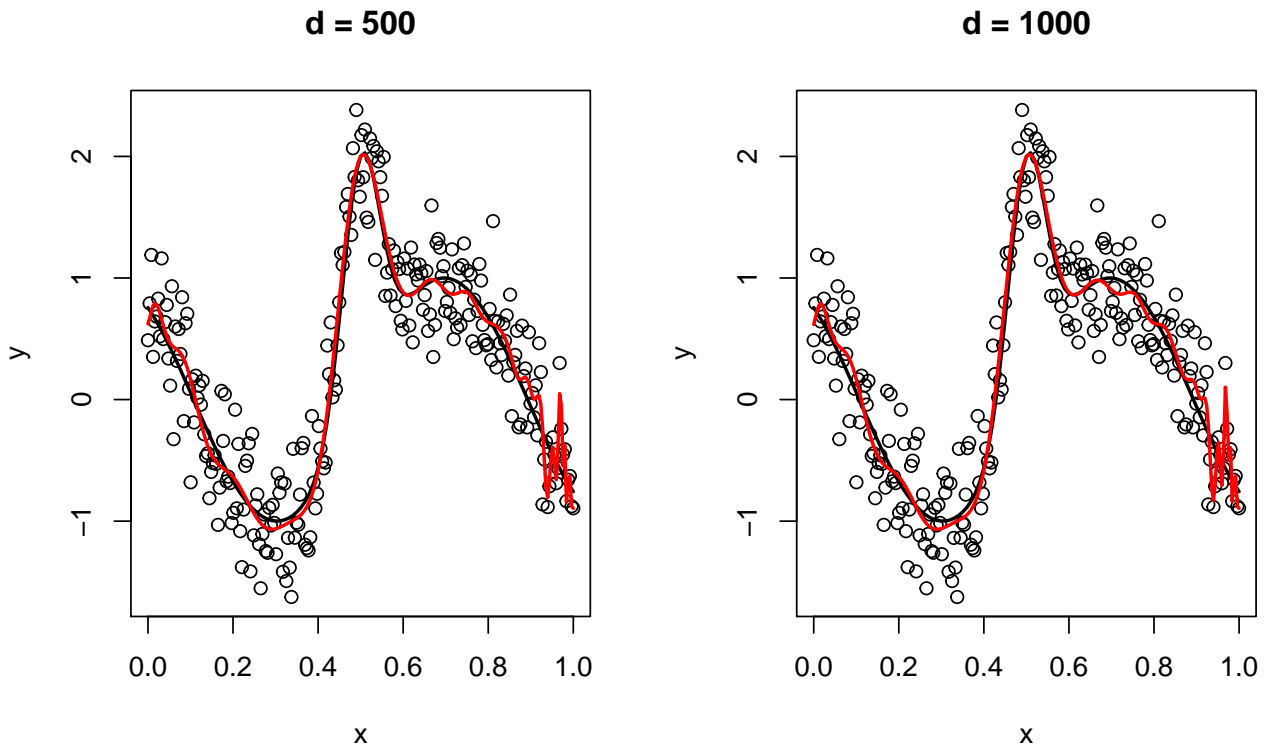
```

par(mfrow=c(1,2))

plot(x, y, main = "d = 500")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit500$fit), lwd = 2, col = "red")

plot(x, y, main = "d = 1000")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit1000$fit), lwd = 2, col = "red")

```



Fazit: ab einem gewissen Polynomgrad kann zwar der wahre Verlauf der Kurve in etwa durch das Polynom nachvollzogen werden, aber dafür steigt das Overfitting und die Schätzung wird, vor allem an den Rändern, instabil. Wie wirkt sich dies auf den Prognosefehler auf den Testdaten aus? Berechne und ordne die die Residuenquadratsummen an:

```
RSSs <- matrix(nrow=11, ncol=2)
RSSs[1,] <- c(fit1$rss, fit1$rsstest)
RSSs[2,] <- c(fit2$rss, fit2$rsstest)
RSSs[3,] <- c(fit3$rss, fit3$rsstest)
RSSs[4,] <- c(fit5$rss, fit5$rsstest)
RSSs[5,] <- c(fit10$rss, fit10$rsstest)
RSSs[6,] <- c(fit20$rss, fit20$rsstest)
RSSs[7,] <- c(fit50$rss, fit50$rsstest)
RSSs[8,] <- c(fit100$rss, fit100$rsstest)
RSSs[9,] <- c(fit250$rss, fit250$rsstest)
RSSs[10,] <- c(fit500$rss, fit500$rsstest)
RSSs[11,] <- c(fit1000$rss, fit1000$rsstest)

colnames(RSSs) <- c("RSS auf Trainingsdaten", "Prädiktive RSS auf Testdaten")
rownames(RSSs) <- c("d = 1", "d = 2", "d = 3", "d = 5", "d = 10", "d = 20", "d = 50",
                    "d = 100", "d = 250", "d = 500", "d = 1000")
```

```
# die folgende Option verhindert, dass große Zahlen als "1.12345e10" dargestellt
# werden.
```

```
options(scipen = 15)
```

```
RSSs
```

##	RSS auf Trainingsdaten	Prädiktive RSS auf Testdaten
## d = 1	197.24743	367.21525
## d = 2	184.53942	341.10052
## d = 3	90.60503	166.31968
## d = 5	68.06513	117.48216
## d = 10	35.54140	64.57532
## d = 20	26.87599	52.13178
## d = 50	24.51925	48.26663
## d = 100	23.81070	49.08493
## d = 250	23.61944	51.17961

## d = 500	23.23753	27052.79191
## d = 1000	23.21455	6547502.25738

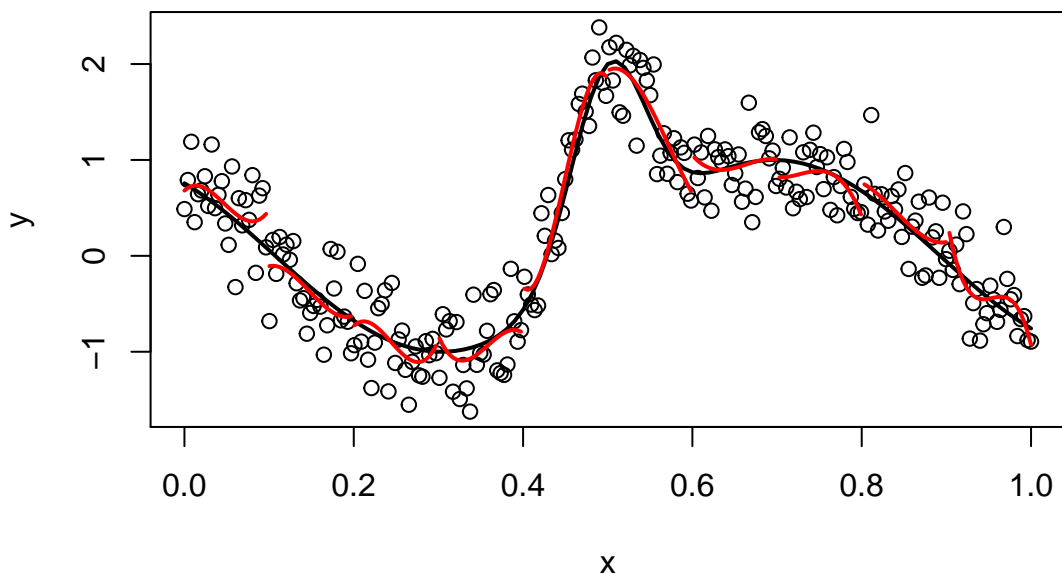
Man sieht: je komplexer das verwendete Modell, desto besser der Fit auf den Trainingsdaten, aber falls zuviel Komplexität zugelassen wird, tritt eine Überanpassung an die Trainingsdaten auf, durch die die Prognosequalität auf den Testdaten sinkt.

Insbesondere sind (gewöhnliche) Polynome global definiert, so dass ein x-Wert links im Wertebereich auch die Schätzung auf der rechten Seite des Wertebereichs beeinflusst. (Die Schätzung einer glatten Kurve ist an den Rändern auch bei allen Glättungsmethoden ungenauer als im Zentrum der Daten, aber aus diesem Grund sind gewöhnliche Polynome dort ganz besonders instabil.) Weiteres Problem von Polynomen: die Designmatrix wird schnell kollinear. (Siehe auch die Warnmeldungen.)

b) Gefittete Werte mit stückweisen Polynomen:

```
ind <- matrix(nrow = 10, ncol = 25)
knots = seq(0, 1, by = 0.1)
for(i in 1:(length(knots)-1) ){
  ind[i,] <- which(x >= knots[i] & x <= knots[i+1])
}

plot(x, y)
curve(f(x), 0, 1, add=T, lwd = 2)
for(i in 1:10){
  lines(x[ind[i,]], lm(y[ind[i,]] ~ poly(x[ind[i,]], 3, raw=T))$fitted.values
        , lwd = 2, col = "red")
}
```



Die Schätzung scheint weniger Artefakte zu haben/weniger rau zu sein. Aber die Polynomstücke haben an den Intervallgrenzen unterschiedliche Werte, so dass die geschätzte Funktion unstetig ist.

- c) siehe Übungsmitschrift
- d) Visualisierung des Einflusses von Splinegrad und Anzahl der Knoten. Verschiedene Spline-Grade für jeweils 13 Knoten, B-Spline-Basis wird konstruiert mit der `bs()`-Funktion aus dem `splines` Package:

```
library(splines)
help(bs)

par(mfrow=c(2,2))
```

```

# Grad 1
fit <- lm(y ~ bs(x, degree = 1, df = 13 + 1))
plot(x, y, main = "Spline-Grad 1")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

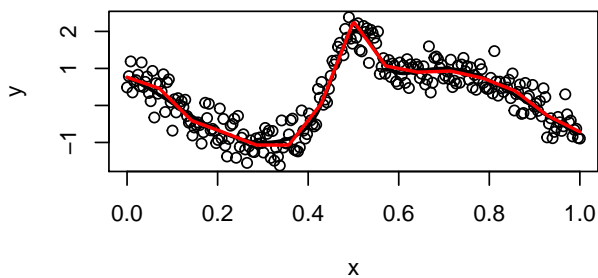
# Grad 2
fit <- lm(y ~ bs(x, degree = 2, df = 13 + 2))
plot(x, y, main = "Spline-Grad 2")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# Grad 3
fit <- lm(y ~ bs(x, degree = 3, df = 13 + 3))
plot(x, y, main = "Spline-Grad 3")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

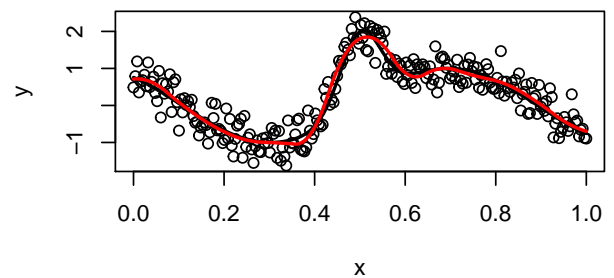
# Grad 10
fit <- lm(y ~ bs(x, degree = 10, df = 13 + 10))
plot(x, y, main = "Spline-Grad 10")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

```

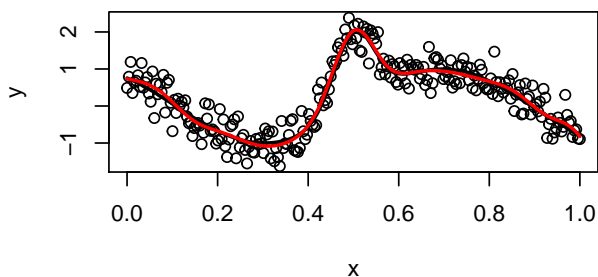
Spline-Grad 1



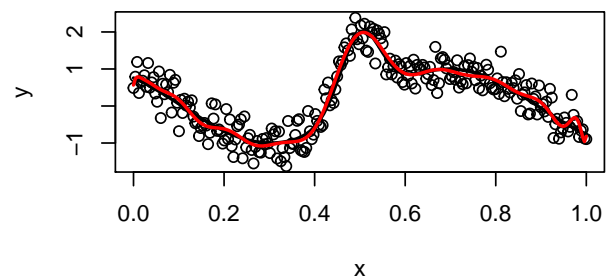
Spline-Grad 2



Spline-Grad 3



Spline-Grad 10



Fazit: für einen Splinegrad von 1 ist die Schätzung noch nicht glatt, auch bei degree = 2 fehlt dem Smoother noch Flexibilität. Ab einem Grad von 3 ist die Schätzung zufriedenstellend glatt, aber mit wachsendem Splinegrad kommt es (bei konstant gehaltener Knotenanzahl) zu Overfitting. In der Praxis wird nahezu immer ein Grad von 3, also kubische Splines, verwendet.

Einfluss der Knotenzahl, für festen Splinegrad von 3 (kubisch):

```

par(mfrow=c(3,3))

# 1 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 1 + 3))
plot(x, y, main = "1 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)

```

```

lines(x, predict(fit), lwd = 2, col = "red")

# 2 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 2 + 3))
plot(x, y, main = "2 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 3 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 3 + 3))
plot(x, y, main = "3 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 5 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 5 + 3))
plot(x, y, main = "5 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 7 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 7 + 3))
plot(x, y, main = "7 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

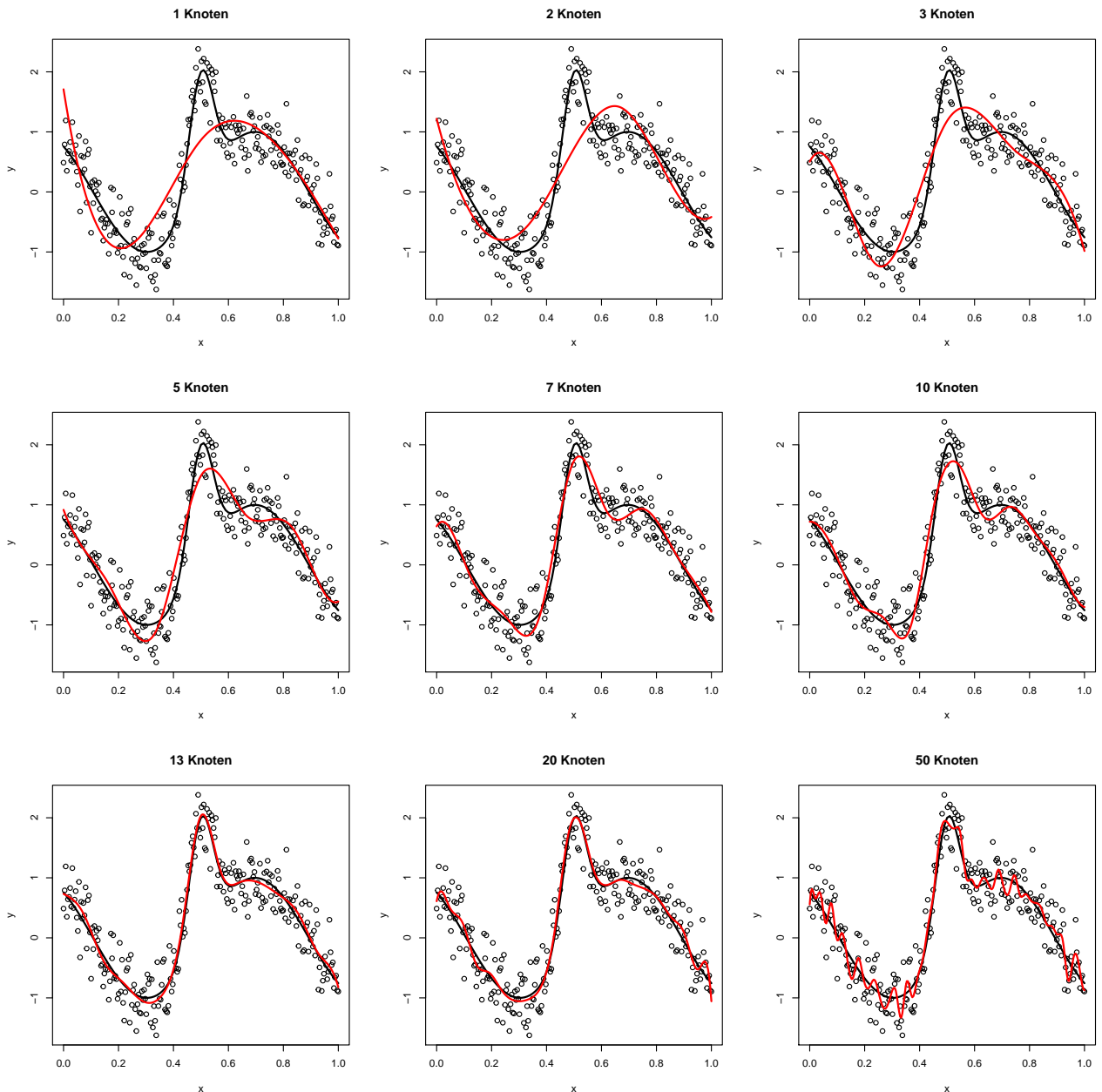
# 10 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 10 + 3))
plot(x, y, main = "10 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 13 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 13 + 3))
plot(x, y, main = "13 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 20 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 20 + 3))
plot(x, y, main = "20 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

# 50 Knoten
fit <- lm(y ~ bs(x, degree = 3, df = 50 + 3))
plot(x, y, main = "50 Knoten")
curve(f(x), 0, 1, add=T, lwd = 2)
lines(x, predict(fit), lwd = 2, col = "red")

```

Fazit: Die Komplexität des Glätters steigt mit der Knotenzahl. Die Wahl einer passenden Knotenzahl ist entscheidend für eine gute Schätzung, aber nicht trivial. Hier wären in etwa 13 Knoten optimal. Möglichkeit zur automatisierten Wahl der Komplexität des Glätters: P-Splines

e) siehe Übungsmitschrift

f) Schätzung durch P-Splines

```
# lade das mgcv-Paket und diverse Hilfeseiten
library(mgcv)
help(gam)
help(s)
help(smooth.terms)
help(plot.gam)
```

Metrische Einflussgrößen können mittels `s()` glatt ins Modell aufgenommen werden. Über das `bs`-Argument kann die Art des Splines gewählt werden. Die Glattheit wird automatisch über GCV bestimmt. Mit dem Argument `'k'` kann die Knotenzahl festgelegt werden.

```
# fitte das Modell mit 13 Knoten
fit <- gam(y ~ s(x, bs = "cr", fx = TRUE, k = 13))
```

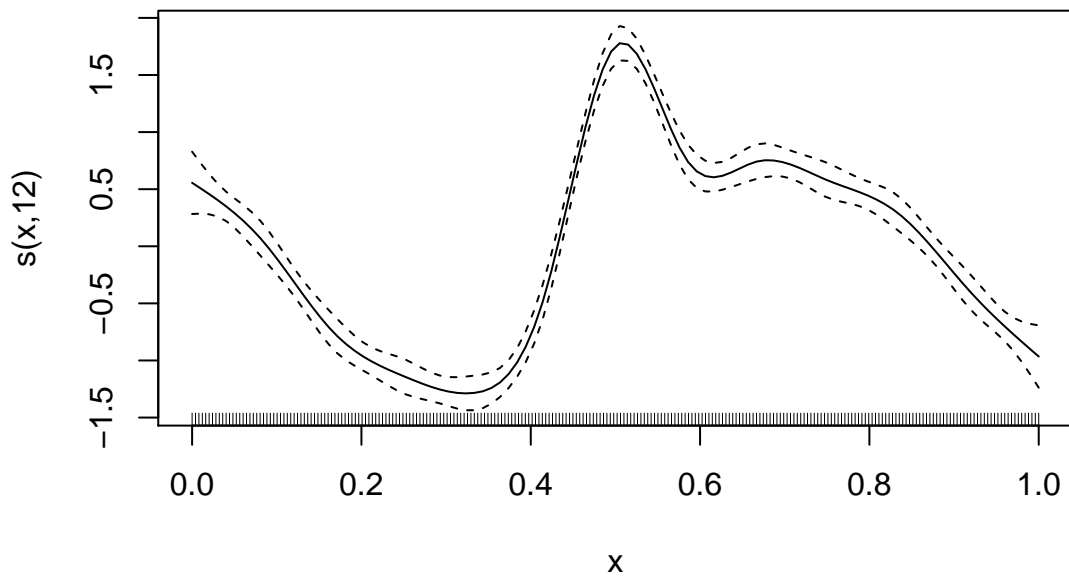
```

#fx = T, weil wir noch nicht penalisieren wollen
summary(fit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x, bs = "cr", fx = TRUE, k = 13)
##
## Parametric coefficients:
##             Estimate Std. Error t value      Pr(>|t|)
## (Intercept)  0.23166    0.02063   11.23 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##      edf Ref.df    F        p-value
## s(x)  12     12 142.3 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.872  Deviance explained = 87.8%
## GCV = 0.11224  Scale est. = 0.1064    n = 250

# die geschätzten Kurven sind natürlich nicht im summary enthalten!
# Plot der geschätzten Kurve:
plot(fit)

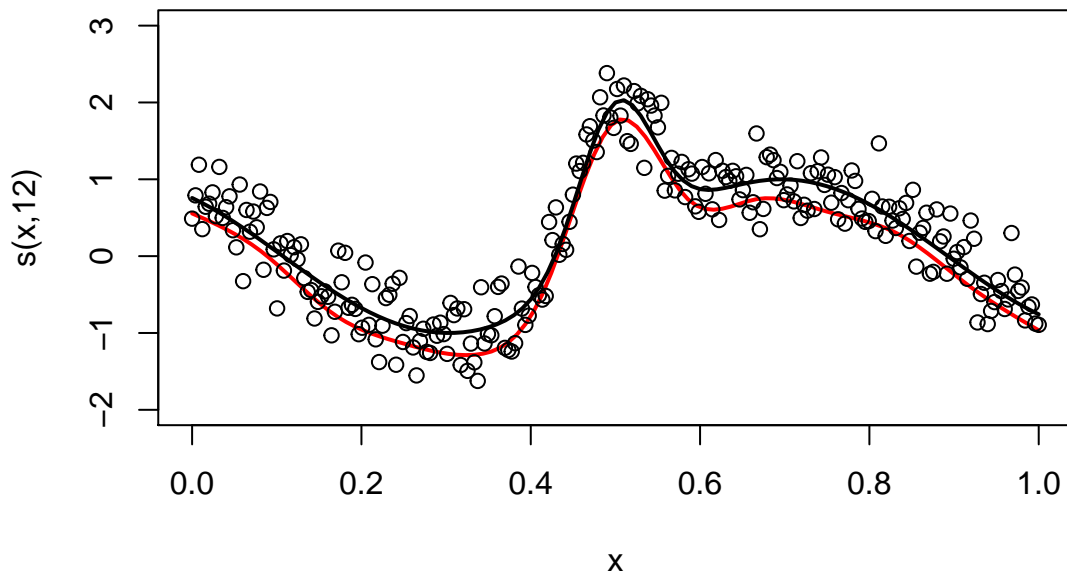
```



```

# nun ein Plot äquivalent zu denen aus Teilaufgabe a)
plot(fit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3))
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

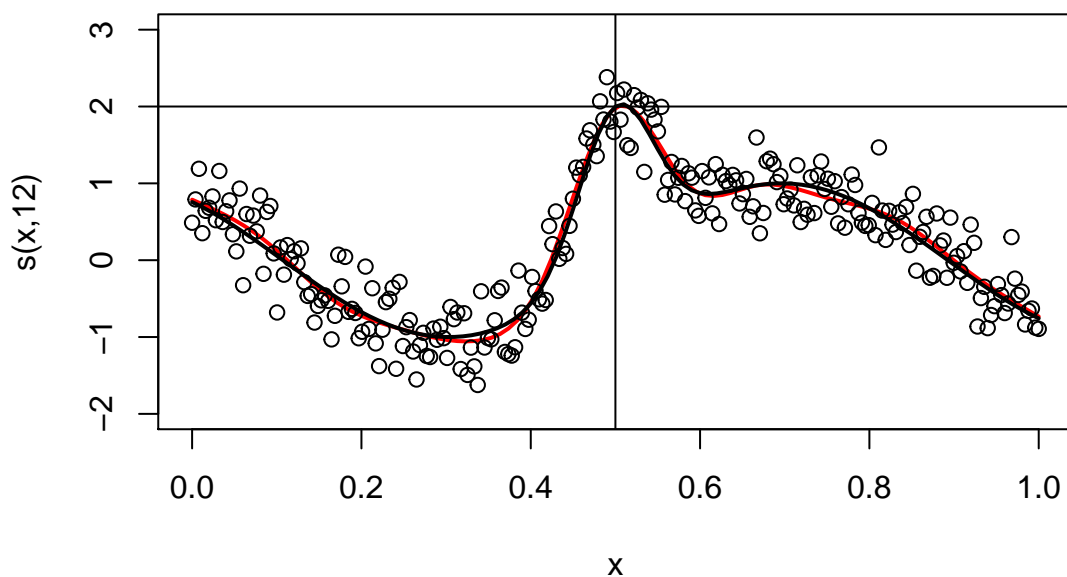
```



Man sieht eindeutig eine Verzerrung, die geschätzte Kurve ist systematisch zu tief. Was lief schief? Glatte Terme in `gam()` werden immer um 0 zentriert ($\sum_{i=1}^n \hat{f}(x_i) = 0$), hier wird aber ein Modell ohne Intercept und mit einer Funktion, die im Mittel bei ungefähr plus 0.25 liegt, gefittet. Wegen der Zentrierung des Glätters kann man in `gam()` nicht einfach ein Modell ohne Intercept fitten. Wenn man nun ein Modell mit Intercept fittet, muss der Intercept noch zum Smoother dazugaddiert werden, (mit der Option `shift`) um die gefitteten Werte zu erhalten.

```
plot(fit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(fit)[1])
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# Beispielinterpretation für den Punkt  $x_i = 0.5$ :
abline(v = 0.5)
# mit Lineal vom Schnittpunkt nach links gehen, um den Wert rauszufinden:
abline(h = 2)
```



Interpretation ist also, dass sich für $x_i = 0.5$ der gefittete Wert $\hat{y}_i = 2$ ergibt. Achtung: falls nur der rohe Plot angegeben ist, ('`plot(fit)`') gibt dieser Schnittpunkt nur den Beitrag von Variable X zum linearen Prädiktor an der gewählten Stelle $x_i = 0.5$ an, nicht aber den gefitteten Wert selbst!

Nun P-Splines mit `gam()`: Achtung: normalerweise verwendet man bei P-Splines eine große Anzahl an Knoten, so circa 20-40. Aus absolut nicht nachvollziehbaren Gründen ist der Default-Wert in `gam()` aber nur 10, was hier nicht ausreichend ist.

```

# Fit mit P-Splines mit dem gam()-Default-Wert von 10 Knoten:
pfit <- gam(y ~ s(x, bs = "ps"))
summary(pfit)

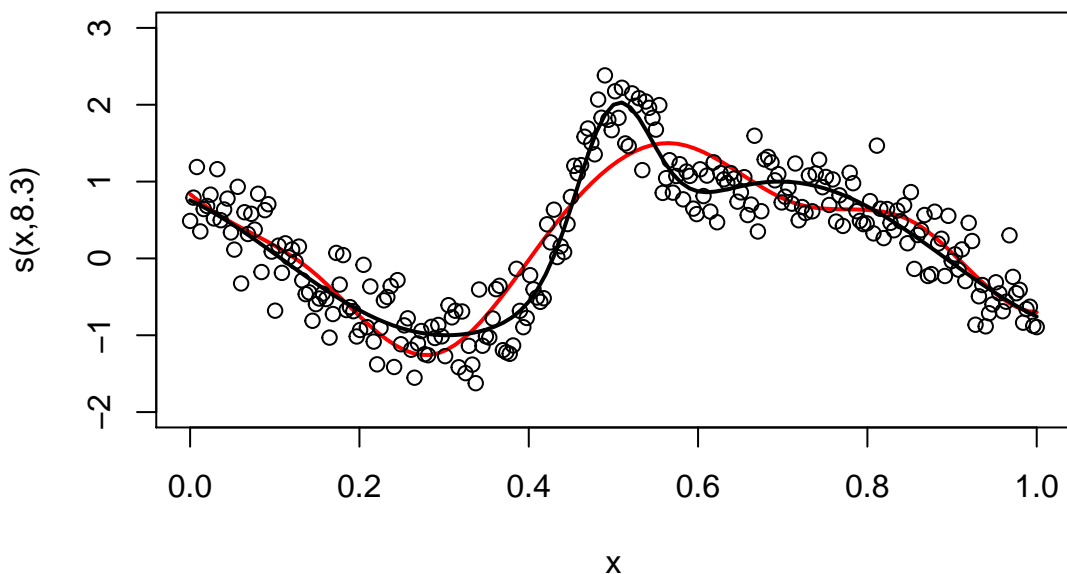
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x, bs = "ps")
##
## Parametric coefficients:
##           Estimate Std. Error t value      Pr(>|t|)
## (Intercept)  0.23166    0.02696   8.594 0.000000000000000108 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df      F        p-value
## s(x)  8.302  8.791 101.8 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.781   Deviance explained = 78.9%
## GCV = 0.18868   Scale est. = 0.18166   n = 250

coef(pfit)

## (Intercept)      s(x).1      s(x).2      s(x).3      s(x).4      s(x).5
##  0.2316602    -0.2560679    -1.0603034    -3.3526153    -0.6516162     0.7343562
##           s(x).6      s(x).7      s(x).8      s(x).9
## -0.9475787    -0.2486944    -2.3067575     0.7344283

# Plot
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
      shift = coef(pfit)[1])
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

```



Der Glätter kann den wahren Kurvenverlauf nicht genau genug erfassen.
 Jetzt dasselbe mit 30 Knoten:

```

pfit <- gam(y ~ s(x, bs = "ps", k = 30))
summary(pfit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x, bs = "ps", k = 30)
##
## Parametric coefficients:
##           Estimate Std. Error t value      Pr(>|t|)
## (Intercept) 0.23166   0.02078   11.15 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df      F        p-value
## s(x) 15.07  17.89 93.11 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.87  Deviance explained = 87.8%
## GCV = 0.11538  Scale est. = 0.10796  n = 250

```

```
coef(pfit)
```

```

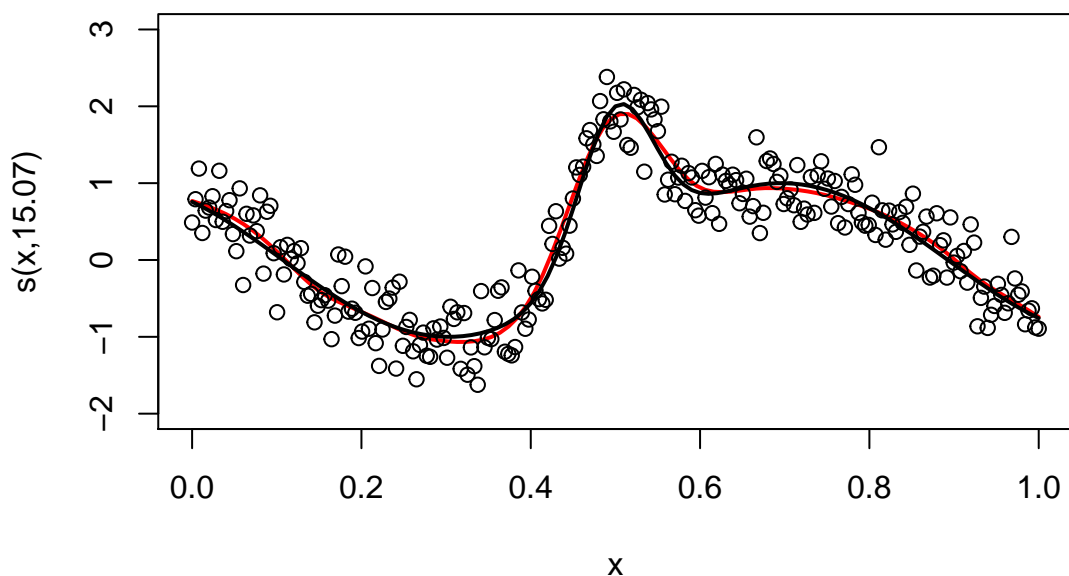
## (Intercept)      s(x).1      s(x).2      s(x).3      s(x).4      s(x).5
## 0.23166019 0.47978958 0.28086760 0.03656845 -0.35033739 -0.78355709
##      s(x).6      s(x).7      s(x).8      s(x).9      s(x).10      s(x).11
## -0.95806699 -1.13600345 -1.34735142 -1.41924575 -1.43821856 -1.31914718
##      s(x).12      s(x).13      s(x).14      s(x).15      s(x).16      s(x).17
## -0.81900980 0.30407198 1.44997908 1.65544274 1.09752577 0.54560851
##      s(x).18      s(x).19      s(x).20      s(x).21      s(x).22      s(x).23
## 0.50651343 0.58812536 0.57396525 0.51493455 0.39572632 0.25093117
##      s(x).24      s(x).25      s(x).26      s(x).27      s(x).28      s(x).29
## 0.03586471 -0.23518865 -0.61406729 -0.84703564 -1.03663331 -1.27757118

```

```

plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
      shift = coef(pfit)[1])
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

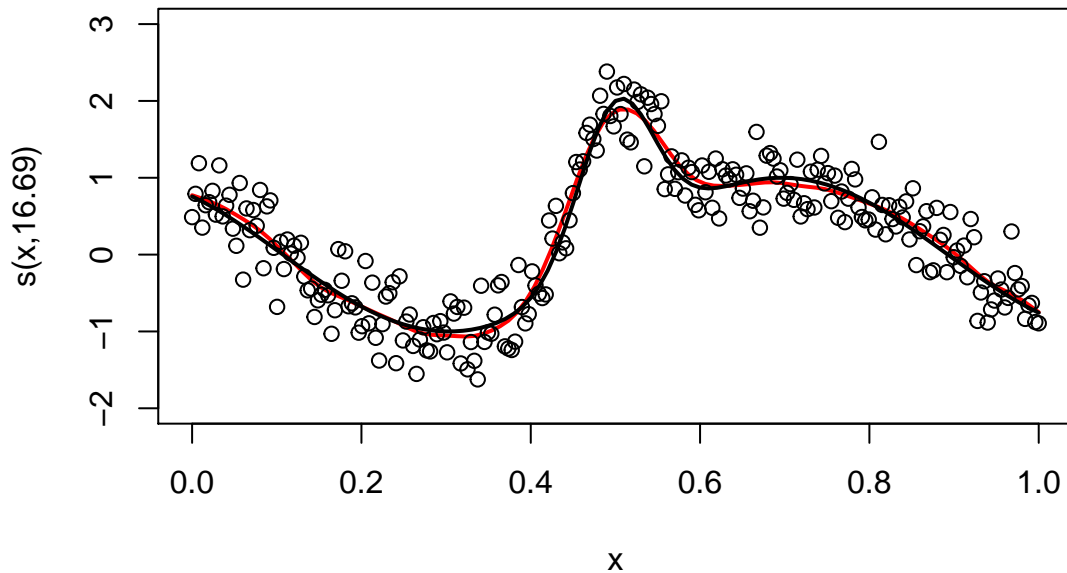
```



Viel besser!

Was passiert, wenn man sehr viele Knoten nimmt? Bleibt der P-Spline-Smoother stabil?

```
pfit <- gam(y ~ s(x, bs = "ps", k = 200))
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1])
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)
```



Trotz viel zu vieler Knoten bleibt der P-Spline-Smoother stabil, da die Penalisierung die überschüssige Komplexität entfernt. Vergleiche auch die effektiven Freiheitsgrade (edf) des Smoothers vs die Zahl der Koeffizienten:

```
summary(pfit)$edf

## [1] 16.69285

length(coef(pfit))

## [1] 200
```

Visualisierung des Einflusses des Glättungsparameters lambda, für k = 30:

```
par(mfrow=c(2,3))

# lambda = 20
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 20)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 20")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# lambda = 6
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 6)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 6")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# lambda = 3
```

```

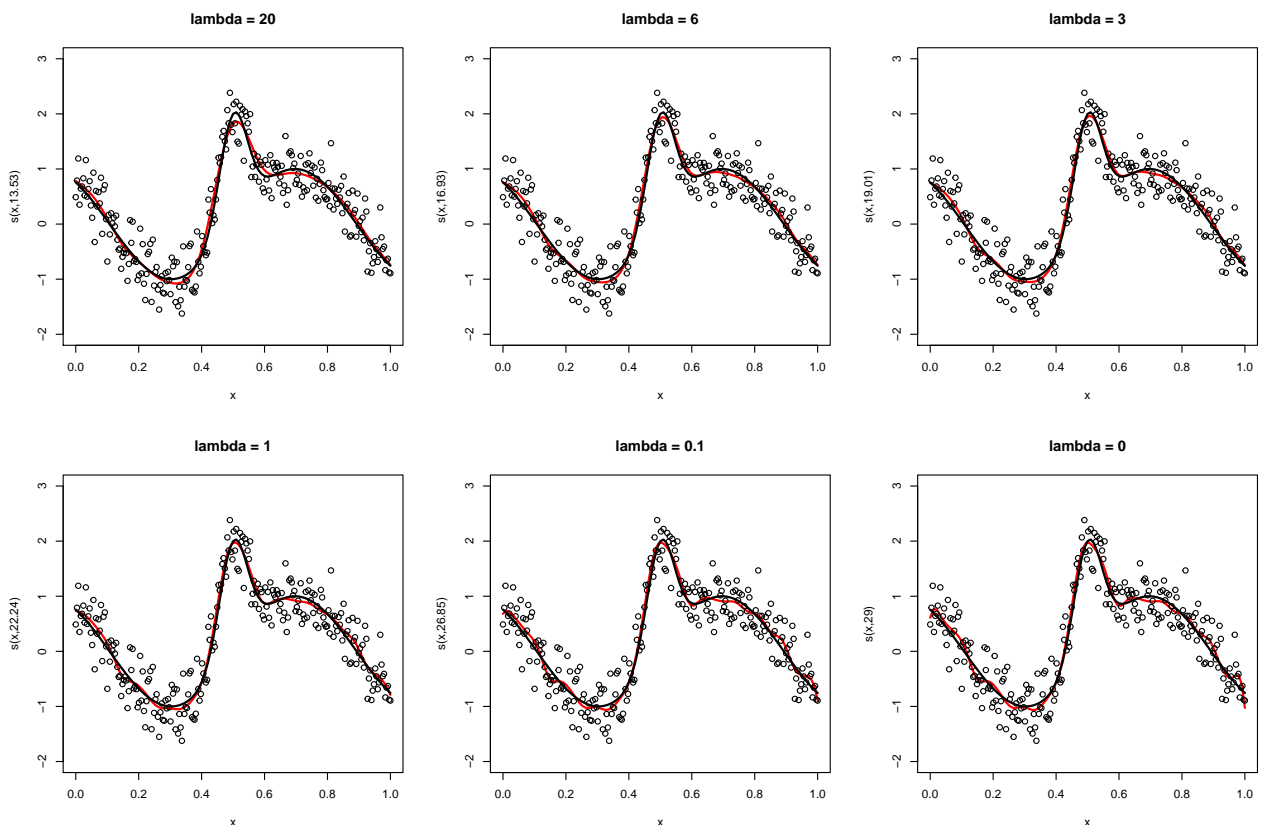
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 3)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 3")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# lambda = 1
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 1)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 1")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# lambda = 0.1
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 0.1)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 0.1")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

# lambda = 0
pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 0)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 0")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

```



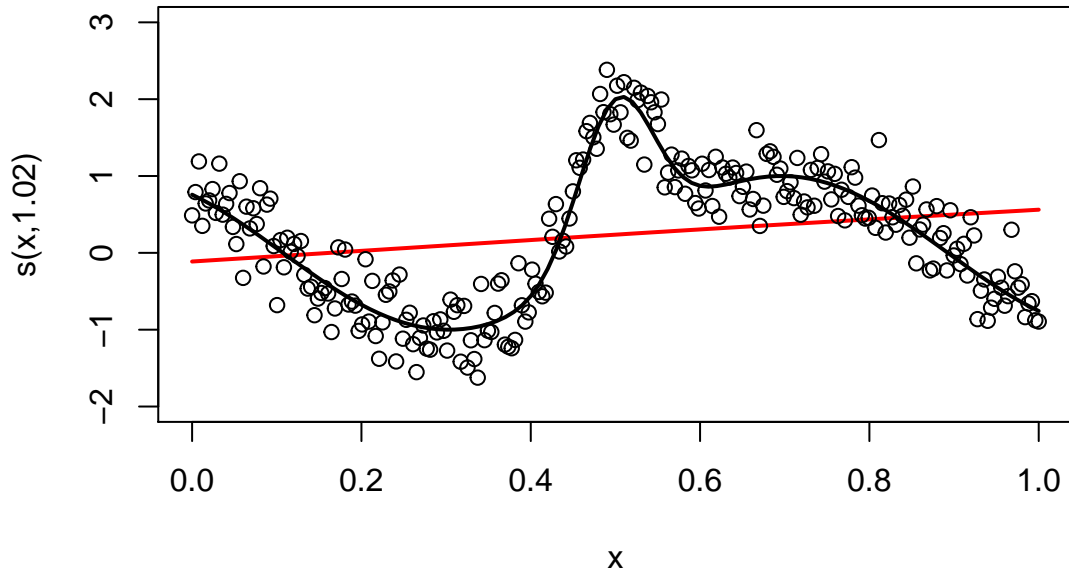
Was passiert, wenn lambda gegen unendlich geht?

```

pfit <- gam(y ~ s(x, bs = "ps", k = 30), sp = 1000000)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 1000000")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)

```

lambda = 1000000

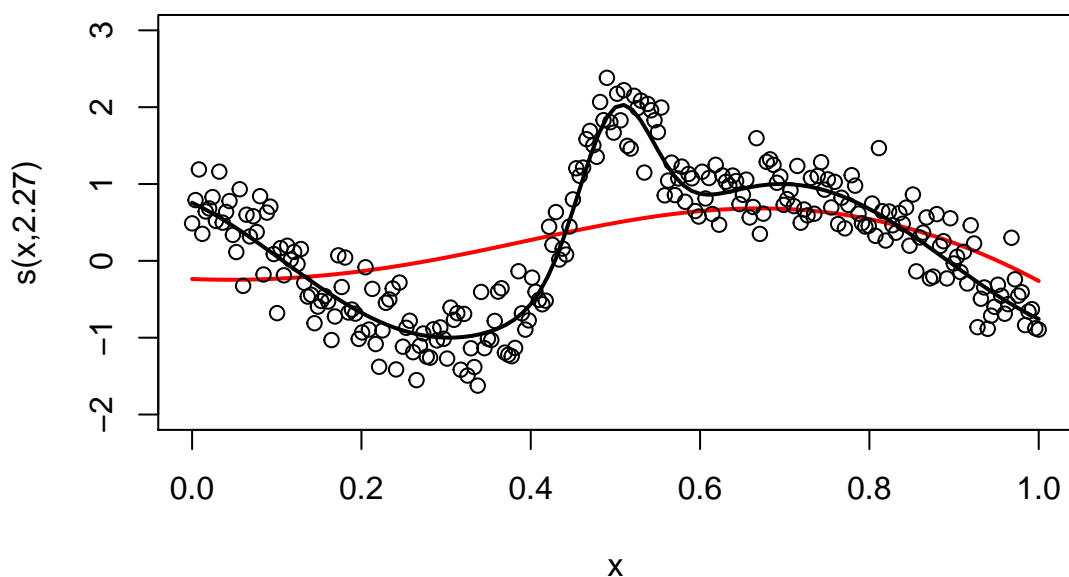


Man erhält eine Gerade! Grund: hier wurden per Default zweite Differenzen verwendet. Wegen der (fast) unendlich starken Penalisierung der zweiten Differenzen entspricht der Schätzer dem Typ von Funktion, für den die zweiten Differenzen gleich Null sind. Man kann zeigen (bitte nicht selber probieren...), dass dies die Klasse der linearen Funktionen ist.

Äquivalent erhält man eine quadratische Funktion, wenn dritte statt zweiter Differenzen betrachtet werden:

```
pfit <- gam(y ~ s(x, bs = "ps", k = 30, m = 3), sp = 10000000)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 10000000")
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)
```

lambda = 10000000



Oder eine konstante Funktion bei ersten Differenzen:

```
pfit <- gam(y ~ s(x, bs = "ps", k = 30, m = 1), sp = 10000000)
plot(pfit, se = FALSE, rug = F, lwd = 2, col = "red", ylim = c(-2,3),
     shift = coef(pfit)[1], main = "lambda = 10000000")
```



```
points(x,y)
curve(f(x), 0, 1, lwd = 2, add = TRUE)
```

lambda = 1000000

