

NLNP Praktikum 9

Robin Baudisch, Merlin Kopfmann, Maximilian Neudert

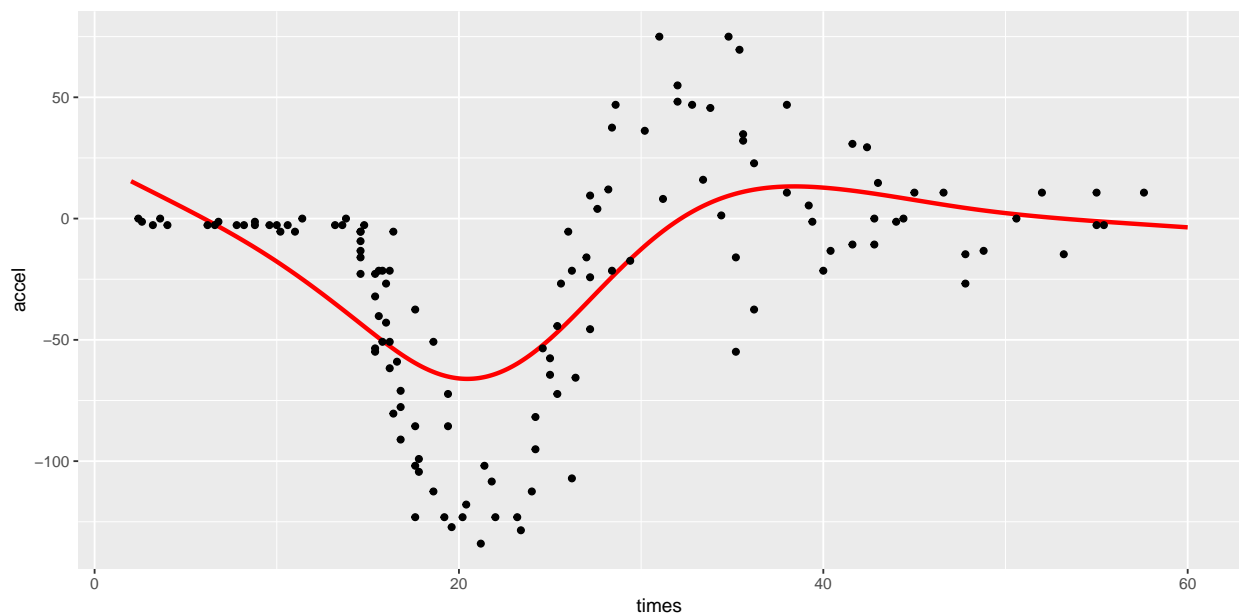
Inhaltsverzeichnis

A1	2
e)	2
f)	2
g)	4
A2	6
a)	6
A3)	8
a)	8
b)	9
c)	10
d)	11
e)	14
f)	14
A4	15

A1**e)**

Wir wählen nun $k = 5$ als effective degrees of freedom und berechnen damit \hat{f} als Regression Smoothing Spline.

```
data = mcycle
RSS = smooth.spline(x = data$times, y = data$accel, df = 5)
fx = seq(2, 60, 0.1)
fy = predict(RSS, fx)$y
ggdf = data.frame(x = fx, y = fy)
gg = ggplot(data = ggdf, mapping = aes(x = x, y = y))
gg = gg + geom_line(color = "red", size = 1.2)
gg = gg + geom_point(data = mcycle, mapping = aes(x = times,
  y = accel), color = "black")
gg = gg + xlab("times") + ylab("accel")
gg
```

**f)**

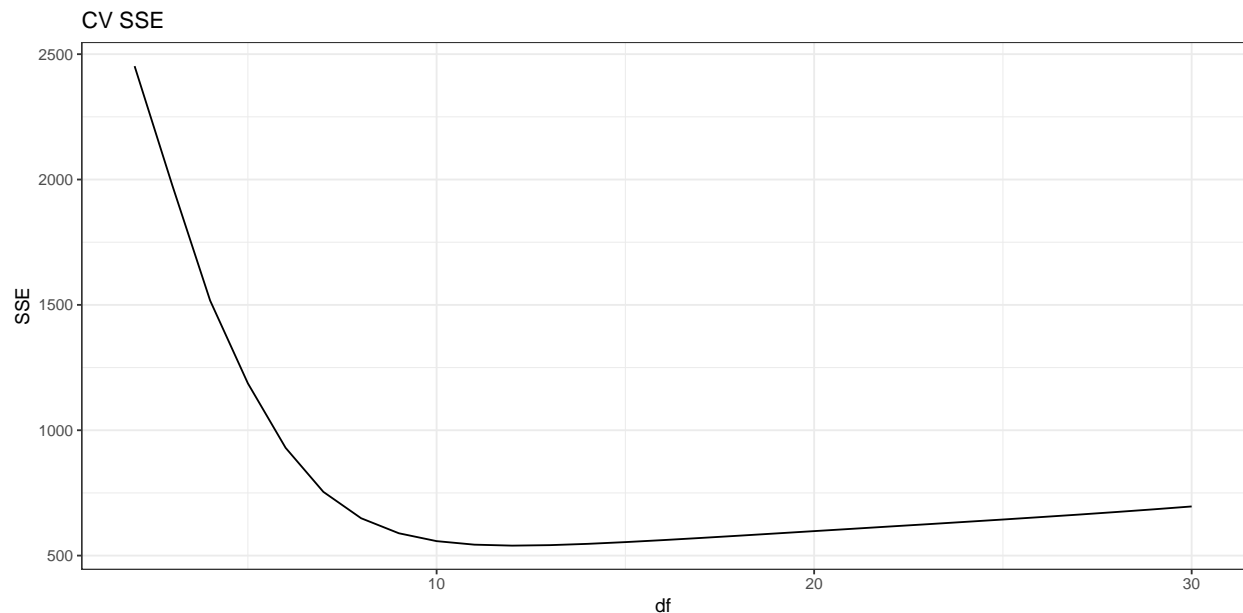
Wir führen im Folgenden eine Parametrisierung für die effective degrees of freedom des smoothing Splines durch. Dabei variieren wir den Wert von 2 bis 30 und untersuchen anschließend anhand einer Grafik, welcher Wert den geringsten Fehler liefert. Den Fehler berechnen wir mit Hilfe einer Leave One Out Cross Validations.

```
my_data <- mcycle
my_data <- my_data[!duplicated(my_data$times), ]
k <- 30
sse_ <- c()
for (j in 2:k) {
  smspl <- smooth.spline(x = my_data$times, y = my_data$accel,
    df = j, cv = TRUE)
  sse_ = append(sse_, smspl$cv.crit)
```

```

}
df_res_ = data.frame(df = 2:k, SSE = sse_)
df_res_ %>% ggplot(., aes(x = df, y = SSE)) + geom_line() + ggtitle("CV SSE") +
  theme_bw()

```



```

smsplDf <- smooth.spline(x = my_data$times, y = my_data$accel,
  df = 10)
lambda = smspl$lambda

```

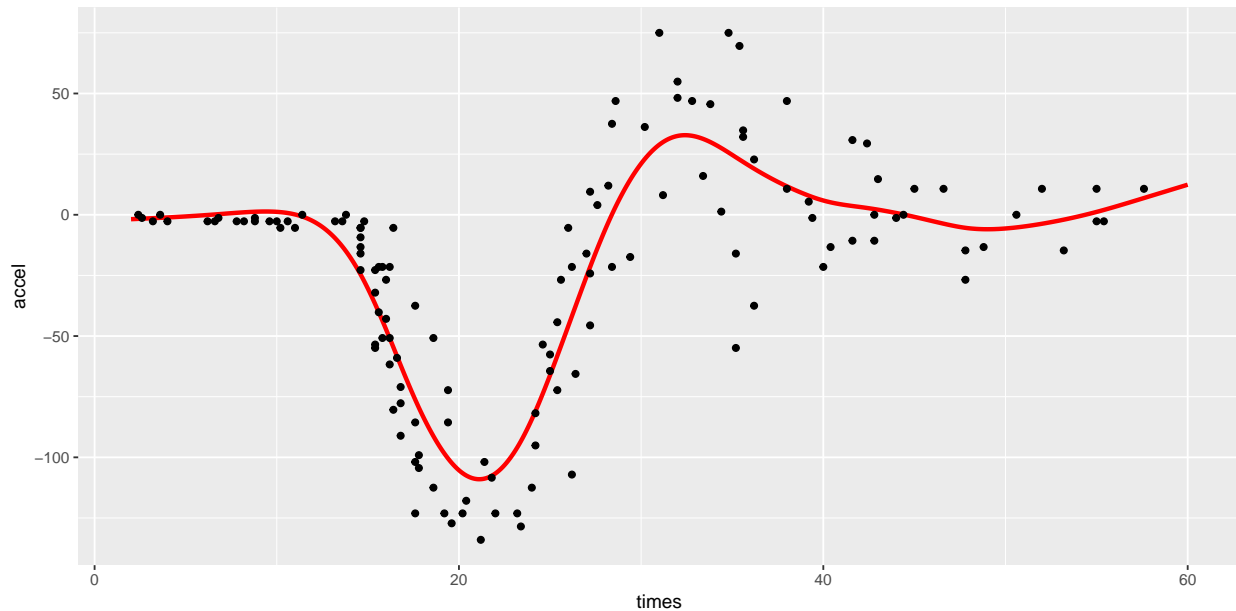
Der optimale Wert für die effektive degrees of freedom ist laut Grafik 10 und damit größer als k . Wir erhalten $\lambda \approx 1.2641699 \times 10^{-6}$.

Wir plotten das Ergebnis.

```

data = mcycle
RSS = smooth.spline(x = data$times, y = data$accel, df = 10)
fx = seq(2, 60, 0.1)
fy = predict(RSS, fx)$y
ggdf = data.frame(x = fx, y = fy)
gg = ggplot(data = ggdf, mapping = aes(x = x, y = y))
gg = gg + geom_line(color = "red", size = 1.2)
gg = gg + geom_point(data = mcycle, mapping = aes(x = times,
  y = accel), color = "black")
gg = gg + xlab("times") + ylab("accel")
gg

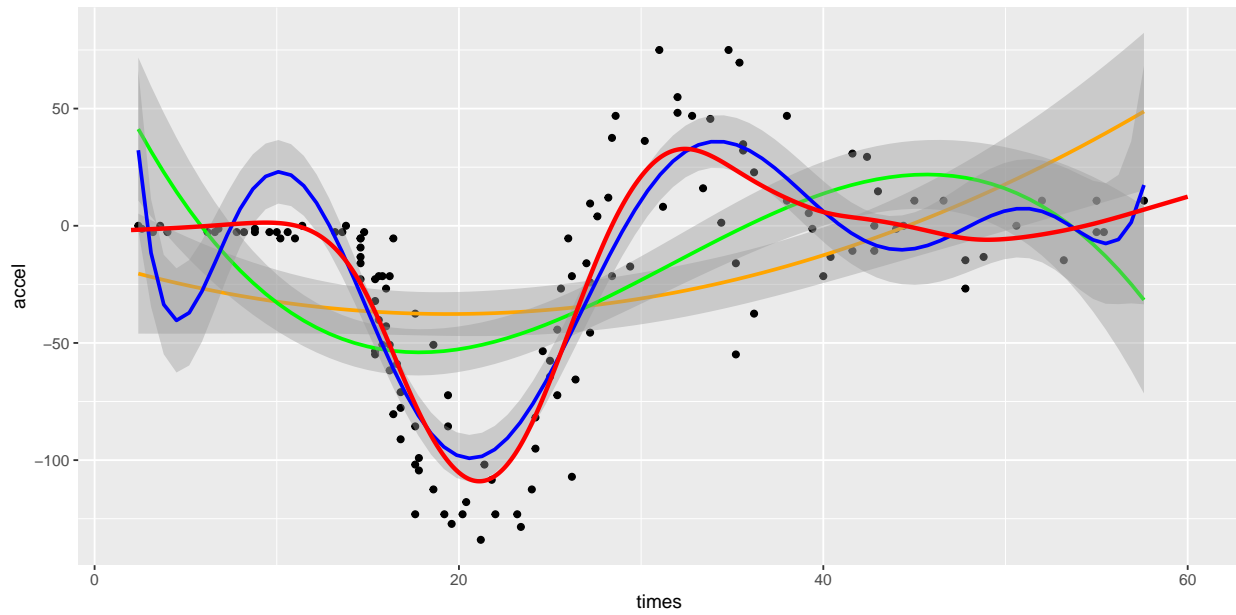
```



g)

Jetzt vergleichen wir den RSS mit den polynomialen Regressionen von Grad 2, 3, 8 aus Praktikum 7.

```
poly2 <- lm(data = mcycle, accel ~ poly(times, 2))
poly3 <- lm(data = mcycle, accel ~ poly(times, 3))
poly8 <- lm(data = mcycle, accel ~ poly(times, 8))
RSS = smooth.spline(x = data$times, y = data$accel, df = 10)
fx = seq(2, 60, 0.1)
fy = predict(RSS, fx)$y
ggdf = data.frame(x = fx, y = fy)
gg = ggplot(data = mcycle, mapping = aes(x = times, y = accel))
gg = gg + geom_point()
gg = gg + geom_smooth(method = "lm", formula = y ~ poly(x, 2),
  color = "orange")
gg = gg + geom_smooth(method = "lm", formula = y ~ poly(x, 3),
  color = "green")
gg = gg + geom_smooth(method = "lm", formula = y ~ poly(x, 8),
  color = "blue")
gg = gg + geom_line(data = ggdf, mapping = aes(x = fx, y = fy),
  color = "red", size = 1.2)
gg
```



- Polynomiale Regression von Grad 2: Orange
- Polynomiale Regression von Grad 3: Grün
- Polynomiale Regression von Grad 8: Blau
- RSS mit $df = 10$: Rot

Das beste Ergebnis nach Sichtprüfung erzielt der Smoothing Spline mit $df = 10$.

A2**a)**Sei λ fest.

$$\hat{f}(x) = \left(\sum_{i=1}^n K_\lambda(x, x_i) y_i \right) \left(\sum_{i=1}^n K_\lambda(x, x_i) \right)^{-1}$$

Der Tricube-Kern ist definiert als

$$K_\lambda(x, x_i) = \begin{cases} \left(1 - \left|\frac{x-x_i}{\lambda}\right|^3\right)^3, & |x - x_i| \leq \lambda \\ 0, & \text{sonst.} \end{cases}$$

Für x_i fest ist $K_\lambda(x, x_i)$ für jedes x_i ohne $|x - x_i| = \lambda$ als Verknüpfung differenzierbarer Funktionen differenzierbar. Folglich bleiben die Punkte

$$|x - x_i| = \lambda \implies x_0 = \lambda + x_i \text{ und } x_0 = x_i - \lambda$$

als kritische Stellen auf Stetigkeit zu prüfen. Der Kern ist offensichtlich stetig in x_0 , da der Wert beidseitig problemlos gegen 0 konvergiert. Wir prüfen die Differenzierbarkeit:

Sei $x_0 = \lambda + x_i$.

$$\lim_{h \rightarrow 0} = \frac{1}{h} \left(\left(1 - \left|\frac{\lambda + x_i - x_i + h}{\lambda}\right|^3\right)^3 - \left(1 - \left|\frac{\lambda + x_i - x_i}{\lambda}\right|^3\right) \right) \quad (1)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(\left(1 - \left|1 + \frac{h}{\lambda}\right|^3\right)^3 - (1 - 1)^3 \right) \quad (2)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(1 - \left(1 + \frac{h}{\lambda}\right)^3 \right)^3 \quad (3)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(1 - \left(1 + 3\frac{h}{\lambda} + 3\frac{h^2}{\lambda^2} + \frac{h^3}{\lambda^3}\right)^3 \right) \quad (4)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(3\frac{h}{\lambda} + 3\frac{h^2}{\lambda^2} + \frac{h^3}{\lambda^3} \right)^3 = 0 \quad (5)$$

und der linksseitige Beweis ($-h$) folgt analog.Sei $x_0 = x_i - \lambda$.

$$\lim_{h \rightarrow 0} = \frac{1}{h} \left(\left(1 - \left| \frac{x_i - \lambda - x_i + h}{\lambda} \right|^3 \right)^3 - \left(1 - \left| \frac{-\lambda + x_i - x_i}{\lambda} \right|^3 \right) \right) \quad (6)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(\left(1 - \underbrace{\left| -1 + \frac{h}{\lambda} \right|}_{<0 \text{ für } h \rightarrow 0} \right)^3 - (1 - 1)^3 \right) \quad (7)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(1 - \left(1 - \frac{h}{\lambda} \right)^3 \right)^3 \quad (8)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(1 - \left(1 - 3\frac{h}{\lambda} + 3\frac{h^2}{\lambda^2} - \frac{h^3}{\lambda^3} \right)^3 \right)^3 \quad (9)$$

$$= \lim_{h \rightarrow 0} \frac{1}{h} \left(-3\frac{h}{\lambda} + 3\frac{h^2}{\lambda^2} - \frac{h^3}{\lambda^3} \right)^3 = 0 \quad (10)$$

und der linksseitige Beweis ($-h$) folgt analog.

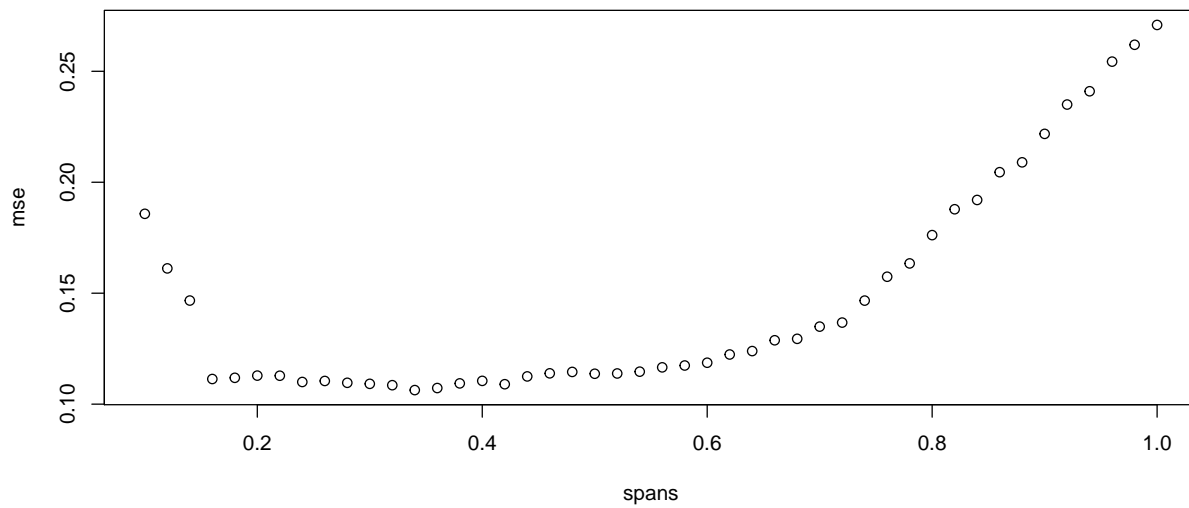
Somit ist $K_\lambda(x, x_i)$ stetig differenzierbar und folglich ist $\hat{f}(x)$ ist Verkettung differenzierbarer Funktionen differenzierbar.

A3)`set.seed(1337)`**a)**

```

data(ethanol)
loessMod10 <- loess(NOx ~ E, data = ethanol, span = 0.1) # 10% smoothing span
loessMod30 <- loess(NOx ~ E, data = ethanol, span = 0.3) # 25% smoothing span
loessMod50 <- loess(NOx ~ E, data = ethanol, span = 0.5)
spans = seq(0.1, 1, 0.02)
# Randomly shuffle the data
data <- ethanol[sample(nrow(ethanol)), ]
# Create 10 equally size folds
folds <- cut(seq(1, nrow(data)), breaks = 5, labels = FALSE)
# Perform 10 fold cross validation
test <- matrix(NA, ncol = 2, nrow = length(spans))
mse1 <- c()
for (span in spans) {
  mse <- c()
  for (j in 1:5) {
    # Segment your data by fold using the which() function
    testIndexes <- which(folds == j, arr.ind = TRUE)
    testData <- data[testIndexes, ]
    trainData <- data[-testIndexes, ]
    # Use the test and train data partitions however you
    # desire...
    fit <- loess(NOx ~ E, data = trainData, span = span,
                 control = loess.control(surface = "direct"))
    y.pred <- predict(fit, newdata = testData$E)
    y.true = testData$NOx
    mse <- append(mse, mean((y.pred - y.true)^2))
  }
  mse1 <- append(mse1, mean(mse))
}
test <- NULL
test$spans <- spans
test$mse <- mse1
test <- as.data.frame(test)
plot(test)

```

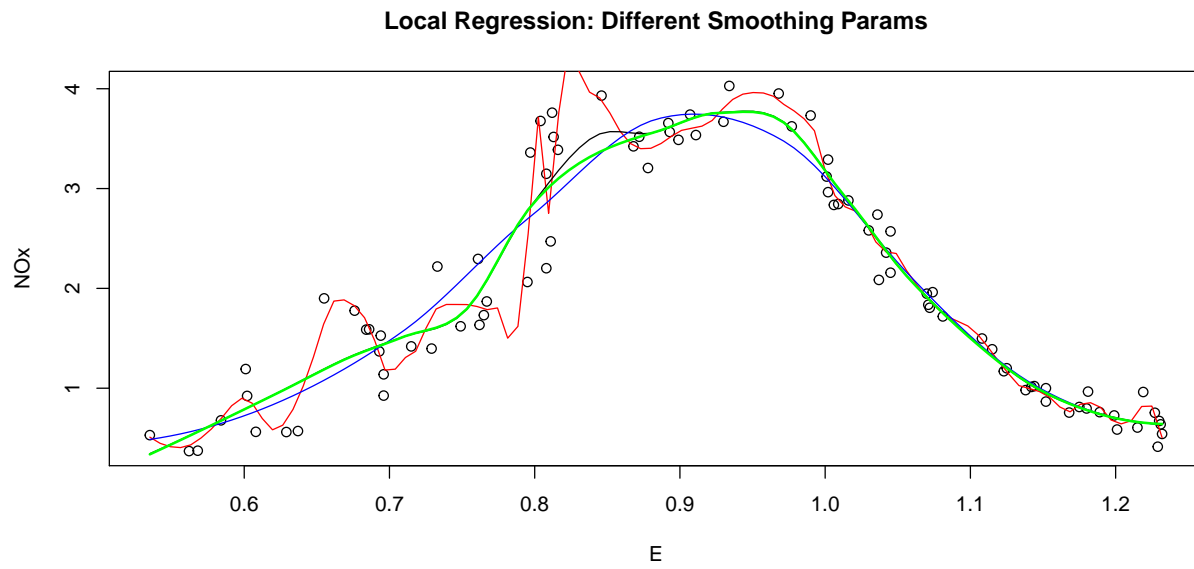
```
opt_spans <- test$spans[test$mse == min(test[, "mse"])]
opt_spans
```

```
[1] 0.34
```

```
loessMod_opt <- loess(NOx ~ E, data = ethanol, span = opt_spans)
```

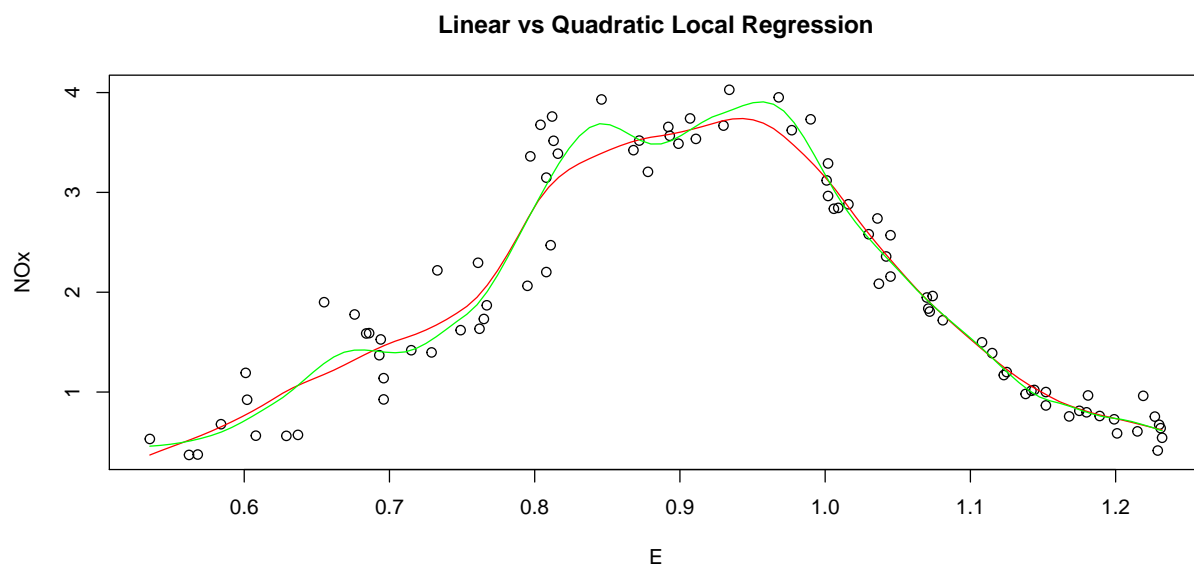
b)

```
tricube_10 <- locfit(NOx ~ lp(E, nn = 0.1), data = ethanol)
tricube_30 <- locfit(NOx ~ lp(E, nn = 0.3), data = ethanol)
tricube_50 <- locfit(NOx ~ lp(E, nn = 0.5), data = ethanol)
tricube_opt <- locfit(NOx ~ lp(E, nn = opt_spans), data = ethanol)
plot(ethanol$NOx, x = ethanol$E, main = "Local Regression: Different Smoothing Params",
     xlab = "E", ylab = "NOx")
lines(tricube_10, col = "red")
lines(tricube_30, col = "black")
lines(tricube_50, col = "blue")
lines(tricube_opt, col = "green", lw = 2)
```



c)

```
lin <- locfit(NOx ~ E, data = ethanol, alpha = 0.2, deg = 1)
quad <- locfit(NOx ~ E, data = ethanol, alpha = 0.2, deg = 2)
plot(ethanol$NOx, x = ethanol$E, main = "Linear vs Quadratic Local Regression",
     xlab = "E", ylab = "NOx")
lines(lin, col = "red")
lines(quad, col = "green")
```



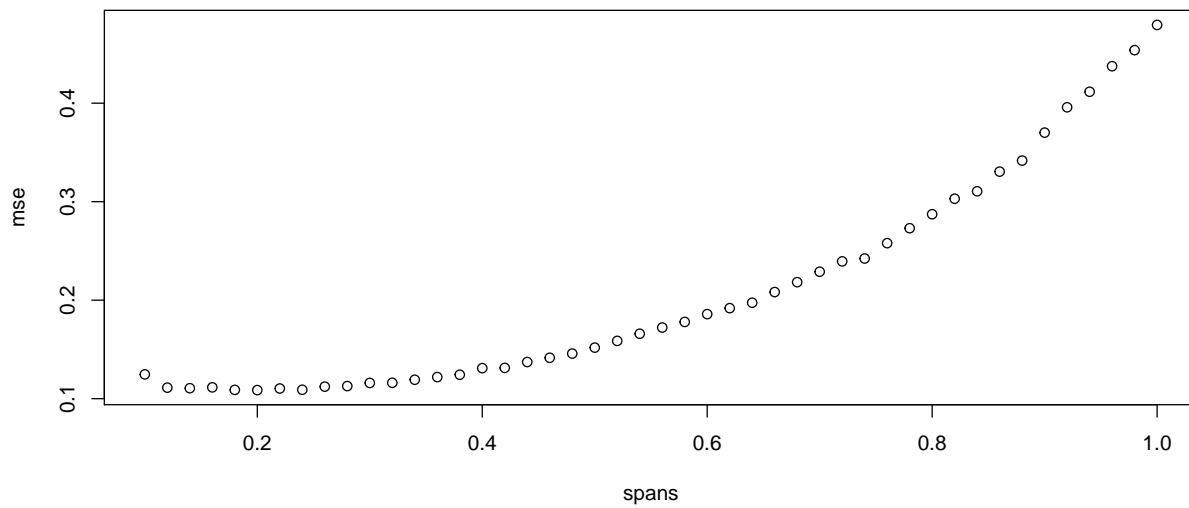
```
lin_pred <- cbind(predict(lin, newdata = c(0.65)), predict(lin,
  newdata = c(0.9)))
```

```
quad_pred <- cbind(predict(quad, newdata = c(0.65)), predict(quad,
  newdata = c(0.9)))
preds <- as.data.frame(rbind(lin_pred, quad_pred), row.names = c("lin",
  "quad"))
colnames(preds) <- c("0.65", "9")
preds
```

	0.65	9
lin	1.146328	3.602708
quad	1.229701	3.562719

d)

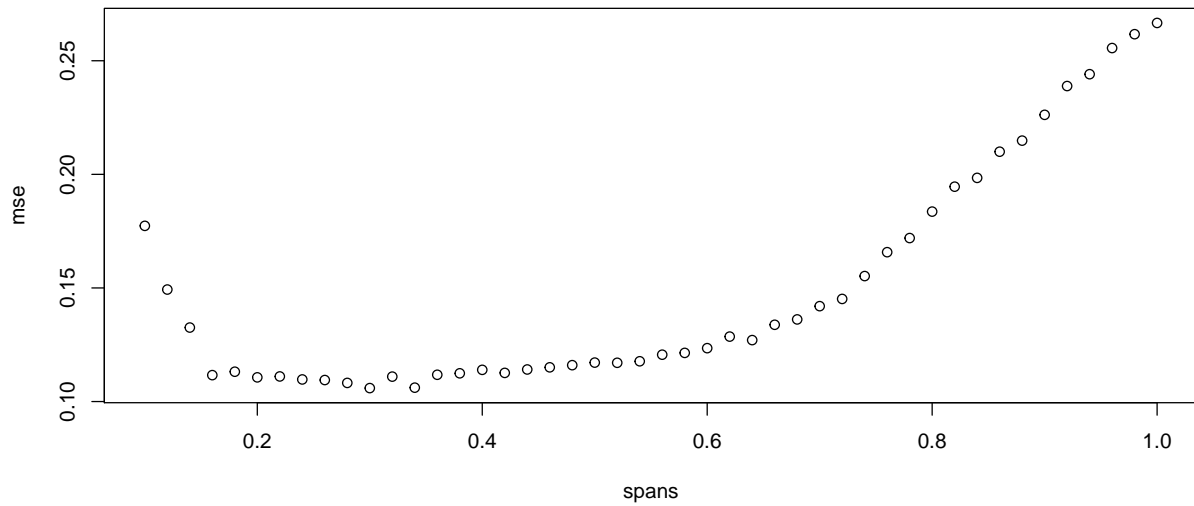
```
mse1 <- c()
for (span in spans) {
  mse <- c()
  for (j in 1:5) {
    # Segment your data by fold using the which() function
    testIndexes <- which(folds == j, arr.ind = TRUE)
    testData <- data[testIndexes, ]
    trainData <- data[-testIndexes, ]
    # Use the test and train data partitions however you
    # desire...
    fit <- locfit(NOx ~ E, data = trainData, alpha = span,
      deg = 1)
    y.pred <- predict(fit, newdata = testData$E)
    y.true = testData$NOx
    mse <- append(mse, mean((y.pred - y.true)^2))
  }
  mse1 <- append(mse1, mean(mse))
}
lin_cv <- NULL
lin_cv$spans <- spans
lin_cv$mse <- mse1
lin_cv <- as.data.frame(lin_cv)
plot(lin_cv)
```



```
opt_lin <- lin_cv$spans[lin_cv$mse == min(lin_cv[, "mse"])]
opt_lin
```

```
[1] 0.2
```

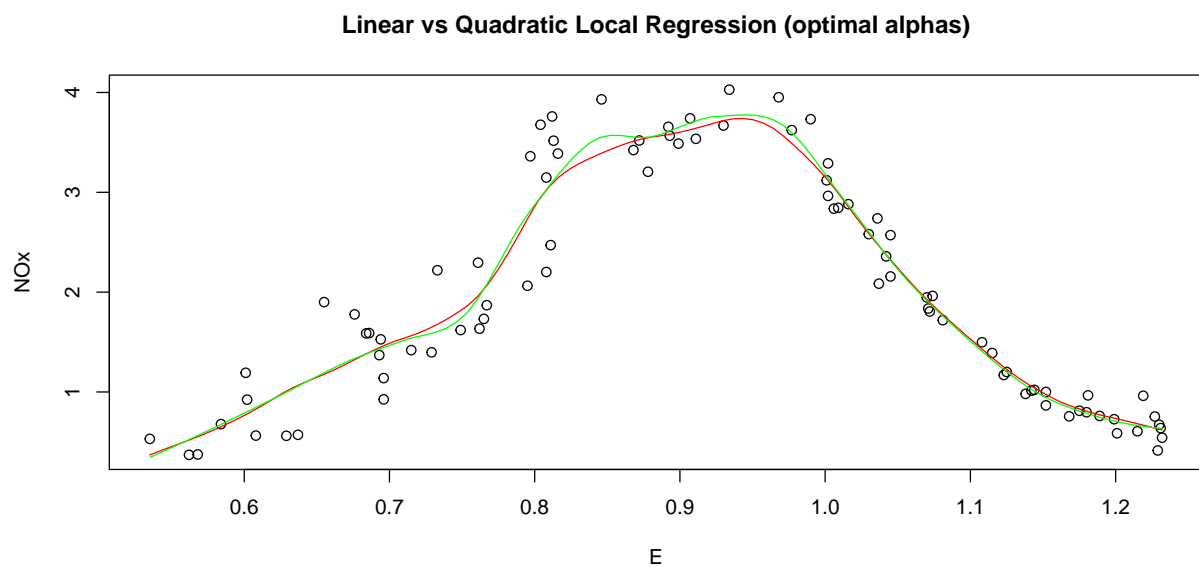
```
mse1 <- c()
for (span in spans) {
  mse <- c()
  for (j in 1:5) {
    # Segment your data by fold using the which() function
    testIndexes <- which(folds == j, arr.ind = TRUE)
    testData <- data[testIndexes, ]
    trainData <- data[-testIndexes, ]
    # Use the test and train data partitions however you
    # desire...
    fit <- locfit(NOx ~ E, data = trainData, alpha = span,
                  deg = 2)
    y.pred <- predict(fit, newdata = testData$E)
    y.true = testData$NOx
    mse <- append(mse, mean((y.pred - y.true)^2))
  }
  mse1 <- append(mse1, mean(mse))
}
quad_cv <- NULL
quad_cv$spans <- spans
quad_cv$mse <- mse1
quad_cv <- as.data.frame(quad_cv)
plot(quad_cv)
```



```
opt_quad <- quad_cv$spans[quad_cv$mse == min(quad_cv[, "mse"])]
opt_quad
```

```
[1] 0.3
```

```
lin_opt <- locfit(NOx ~ E, data = ethanol, alpha = opt_lin, deg = 1)
quad_opt <- locfit(NOx ~ E, data = ethanol, alpha = opt_quad,
  deg = 2)
plot(ethanol$NOx, x = ethanol$E, main = "Linear vs Quadratic Local Regression (optimal alpha)",
  xlab = "E", ylab = "NOx")
lines(lin_opt, col = "red")
lines(quad_opt, col = "green")
```



e)

- Größte Varianz: Lokal quadratisch
- Kleiner Bias: Lokal quadratisch
- Kleinste Varianz: K-Nearest Neighbor
- Größer Bias: K-Nearest Neighbor

und Lokal lineare Regression liegt dazwischen, was in Anbetracht des Bias-Variance-Tradeoff Sinn ergibt, da die Verfahren alle gleich sind mit K-Nearest Grad 0, Lokal Linear Grad 1 und Lokal Quadratisch Grad 2. Das heißt, dass Lokal quadratisch am besten fitten kann, während K-Nearest am meisten glättet.

f)

```
sum(residuals(loessMod10)^2)
```

```
[1] 3.382258
```

```
sum(residuals(loessMod30)^2)
```

```
[1] 7.18373
```

```
sum(residuals(loessMod50)^2)
```

```
[1] 9.168455
```

```
sum(residuals(loessMod_opt)^2)
```

```
[1] 7.369065
```

```
sum(residuals(tricube_10)^2)
```

```
[1] 3.442665
```

```
sum(residuals(tricube_30)^2)
```

```
[1] 7.093455
```

```
sum(residuals(tricube_50)^2)
```

```
[1] 9.067521
```

```
sum(residuals(tricube_opt)^2)
```

```
[1] 7.358256
```

```
sum(residuals(lin_opt)^2)
```

```
[1] 7.395092
```

```
sum(residuals(quad_opt)^2)
```

```
[1] 7.093455
```

A4

Der Gauß Kernel ist definiert als

$$K_\lambda(x, x_i) = \frac{1}{\sqrt{2\pi\lambda}} \exp\left(-\frac{(x - x_i)^2}{2\lambda^2}\right) \quad (11)$$

Der Kernel gewichtet jeden Datenpunkt zusätzlich zu Least Squares

- $\lambda \rightarrow \infty$: Große Bandbreite (über alle Daten). Der Gauß Kernel konvergiert gegen 0, der Abstand zwischen x_j und den x_i bekommt einen kleinen Skalar und führt deshalb dazu, dass der Wert \hat{y}_i dem der linearen Regression $\hat{b}_0 + \hat{b}_1 x_i$ entspricht.
- $\lambda \rightarrow 0$: Geringe Bandbreite (nur der Punkt selbst). Der Gauß Kernel konvergiert gegen 0, der Abstand zwischen x_j und den x_i bekommt einen großen Skalar und führt deshalb dazu, dass der Wert \hat{y}_i gegen y_j konvergiert und somit interpoliert.