```
#> Warning: package 'ggplot2' was built under R version 3.5.3
#> Warning: package 'tidyr' was built under R version 3.5.3
```

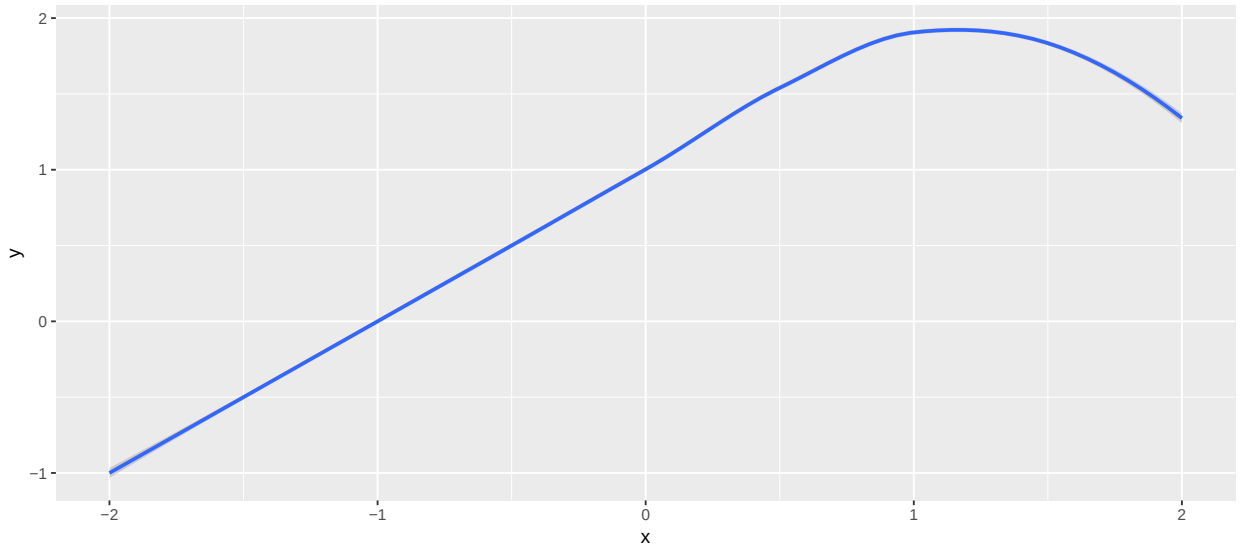
Praktikum 7

Aufgabe 1

Zuerst berechnen wir die Punkte

```
h1 = function (X) {
  return(X)
}
h2 = function (X) {
  ind = as.numeric(X >= 1)
  return((X-1)^2 * ind)
}
Y = function (X) {
  y = 1 + h1(X) - 2 * h2(X)
  return(y)
}
y1 = Y(-2)
y2 = Y(2)
```

Wir erhalten $Y(-2) = -1$ und $Y(2) = 1$. Skizzieren ist nun relativ simpel, da die Funktion von -2 bis 1 linear ist mit Punkten $(-2, -1)$ und $(0, 1)$. Danach haben wir eine negative quadratische Funktion mit Punkt $(2, 1)$ welche im Punkt $(1, 2)$ glatt ist und wir somit nur mit etwas schwung die gerade in eine Kurve skizzieren müssen, die dann in $(2, 1)$ endet. Zum Vergleich ein maschineller Plot:



Aufgabe 2

Der erste Term

$$\sum_{i=1}^{n}(y_i - g(x_i))^2$$

ist der Least Squares Term. Dieser wird minimal, wenn die Kurven die Trainingspunkte interpolieren. Der zweite Term

$$\lambda \int_a^b \left| g^{(3)}(x) \right|^2 \, dx$$

ist ein der Glättungsterm (Misst die ''Kurvigkeit der Funktion'') mit Glättungsparameter λ, der als Strafterm den Funktionswert für starke Veränderungen erhöht. Die Ableitung einer Funktion ist ein Maß für die''Veränderung" einer Funktion. Das heißt, je höher die Ableitung, desto stärker misst man Veränderungen. λ ist dann das Gewicht für den Strafterm. Erhöht man den Wert des Strafterms für die Veränderung, so steigt der Trainingsfehler (Die Funktion wird glatter und interpoliert weniger).

$\hat{g}_2$ lässt kurvigere Funktionen als $\hat{g}_1$ zu und kann somit besser interpolieren (Kleinerer Trainingsfehler). Für $\lambda \to \infty$ wird der Strafterm maximiert und nur noch eine maximal glatte Funktion zugelassen, sprich die Zielfunktion strebt gegen eine Gerade. Für $\lambda \to 0$ wird der Strafterm minimiert, somit ignoriert und folglich strebt die Funktion gegen die Interpolation.

Somit erhalten wir:

a) $\hat{g}_1$ hat im Allgemeinen einen kleineren Trainingsfehler, da die Zielfunktion langsamer sich einer Geraden annähert.

b) $\hat{g}_2$ hat im Allgemeinen einen kleineren Testfehler, da starke Schwankungen schneller glatt gebügelt werden.

c) $\hat{g}_2$ hat im Allgemeinen einen kleineren Trainingsfehler, da stärkere Interpolation.

d) $\hat{g}_1$ hat im Allgemeinen einen kleineren Testfehler, da schwächere Interpolation.

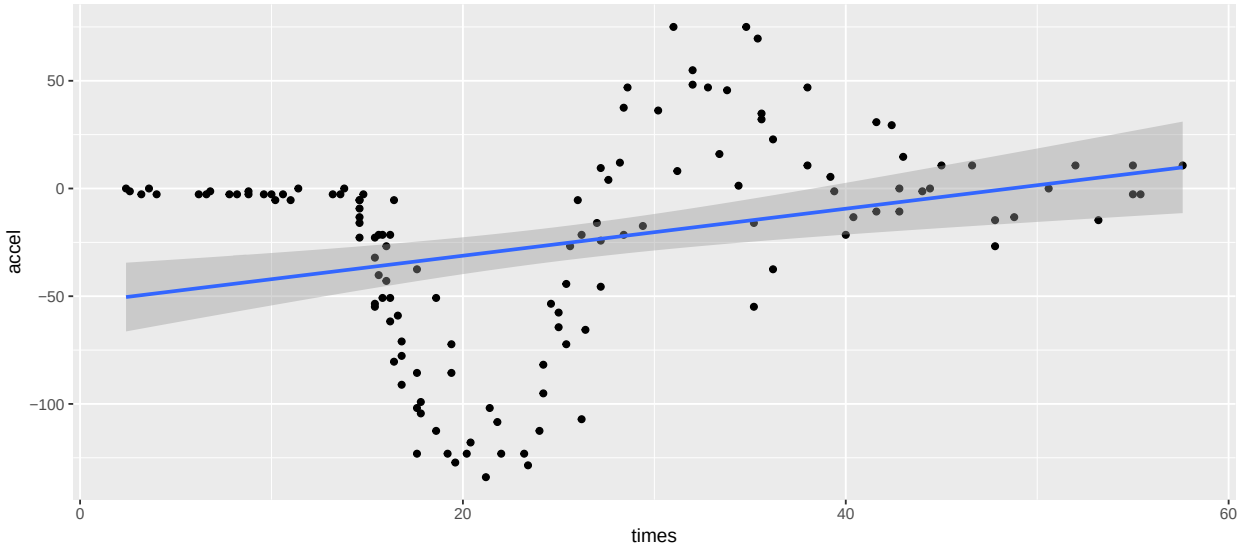
Aufgabe 3

```
df = mcycle
```

a)

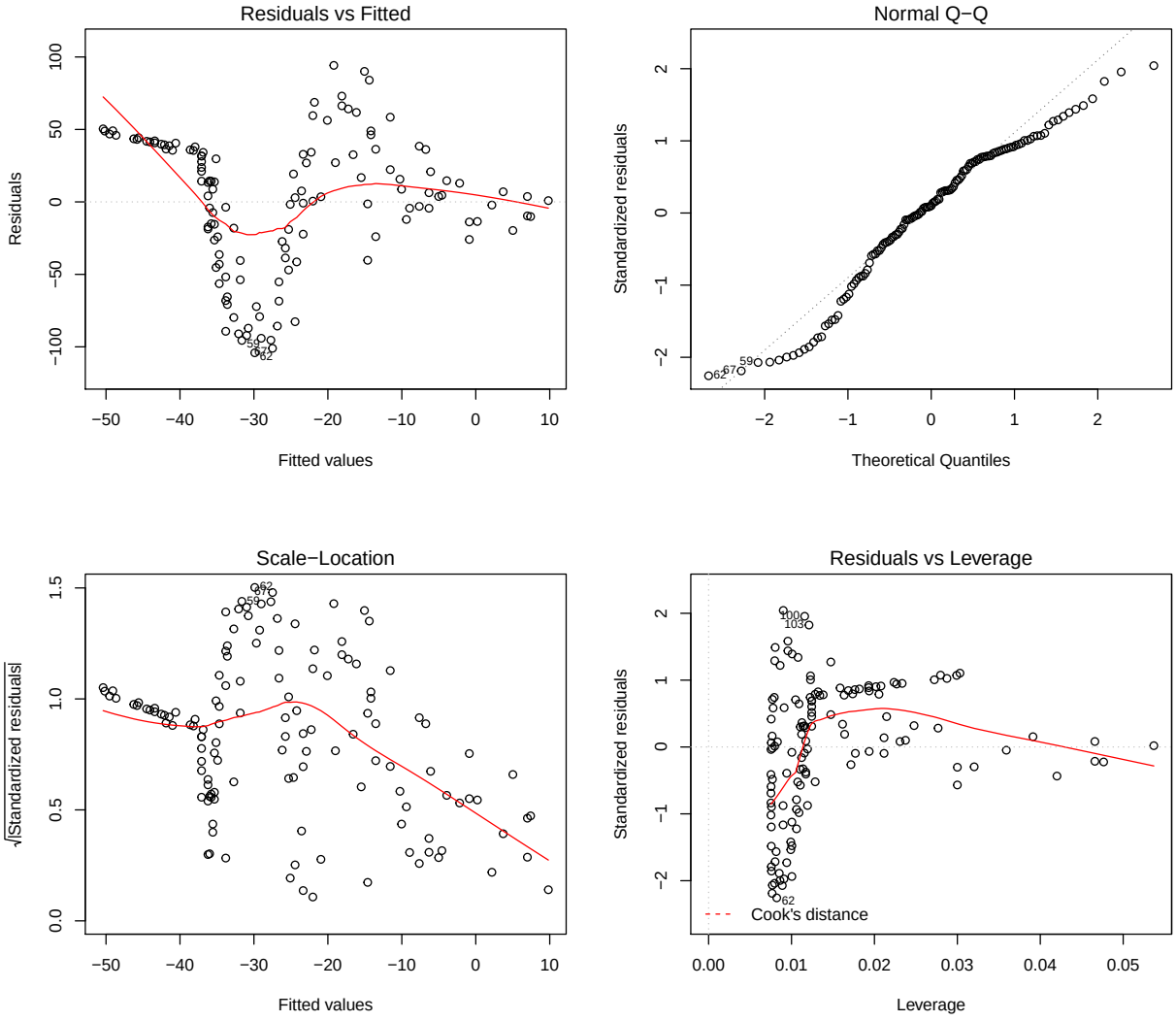
Zuerst schauen wir uns den Plot mit linearer Regression an.

```
gg = ggplot(
  data = df,
  mapping = aes(
    x = times,
    y = accel
  )
)
gg = gg + geom_point()
gg + geom_smooth(
  method = 'lm',
  formula = 'y ~ x'
)
```



Dies sieht nicht so passend aus. Nun betrachten wir die diagnostischen Plots:

```
lm = lm(accel~times, df)
par(mfrow=c(2,2))
plot(lm)
```



Der Scatterplot lässt vermuten, dass die Datenpunkte nicht linear verteilt sind (Modellverletzung). Dies erkennt man auch in den diagnostischen Plots, da die Residuen normalverteilt sein sollten, was sie aber nicht sind (Modellverletzung).

b)

Sei P_n die Menge aller Polynome vom Grad n, also:

$$P_n = \{a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n\}, \quad a_i \in \mathbb{R}$$

Seien $\hat{g}_2$ und $\hat{g}_3$ die Regressionsfunktionen für die polynomiale Regression von Grad 2 und Grad 3. Dann sind unsere Modelle definiert durch:

$$\hat{g}_2 = \arg\min_{g \in P_2} \sum_{i=1}^{n} (y_i - g(x_i))^2 \tag{1}$$

$$\hat{g}_3 = \arg\min_{g \in P_3} \sum_{i=1}^{n} (y_i - g(x_i))^2 \tag{2}$$

Da ein Polynom von Grad n genau $n+1$ freie Parameter hat, hat g_2 3 freie Parameter und g_3 4 freie Parameter.

```r
# models
p2 = lm(accel~poly(times, 2, raw=TRUE), data=df)
p3 = lm(accel~poly(times, 3, raw=TRUE), data=df)
# manual function for prediction
g2 = function(x) {
  g = p2$coefficients[1]
  g = g + p2$coefficients[2] * x
  g = g + p2$coefficients[3] * x^2
  return(g)
}
g3 = function(x) {
  g = p3$coefficients[1]
  g = g + p3$coefficients[2] * x
  g = g + p3$coefficients[3] * x^2
  g = g + p3$coefficients[4] * x^3
  return(g)
}
# times to check
t = c(10, 20, 30, 40)
# manual prediction
val2 = g2(t)
val3 = g3(t)
# prediction using predict
df.t = data.frame(times = t)
pred2 = predict(p2, df.t)
pred3 = predict(p3, df.t)
# Check: Calculate the difference
d2 = val2 - as.vector(pred2)
d3 = val3 - as.vector(pred3)
# round for visual
val2 = round(val2, 2)
val3 = round(val3, 2)
```

Wir erhalten als Ergebnis

t	10	20	30	40
$g_2(t)$	-32.37	-37.64	-31.04	-12.57
$g_3(t)$	-32.79	-52.68	-23.16	13.71

Zur Überprüfung haben wir die Differenzen berechnet und erhalten

t	10	20	30	40
$g_2(t)$ – predict	0	0	0	0
$g_3(t)$ – predict	0	0	0	0

Um zu überprüfen, ob ein lineares Modell ausreichen würde, führen wir einen Anpassungstest auf den Residuen durch. Als Testmethode nehmen wir Shapiro-Wilk.

```
lm = lm(accel~times, data=df)
sw = shapiro.test(lm$residuals)
pval = round(sw$p.value, 5)
sw
#>
#>  Shapiro-Wilk normality test
#>
#> data:  lm$residuals
#> W = 0.96155, p-value = 0.0008342
```

Wir erhalten einen p-Wert von 8.3×10^{-4} dieser ist deutlich kleiner als `0.05` somit können wir annehmen, dass die Residuen der linearen Regression nicht normalverteilt sind und folgich ist ein lineares Modell nicht ausreichend.

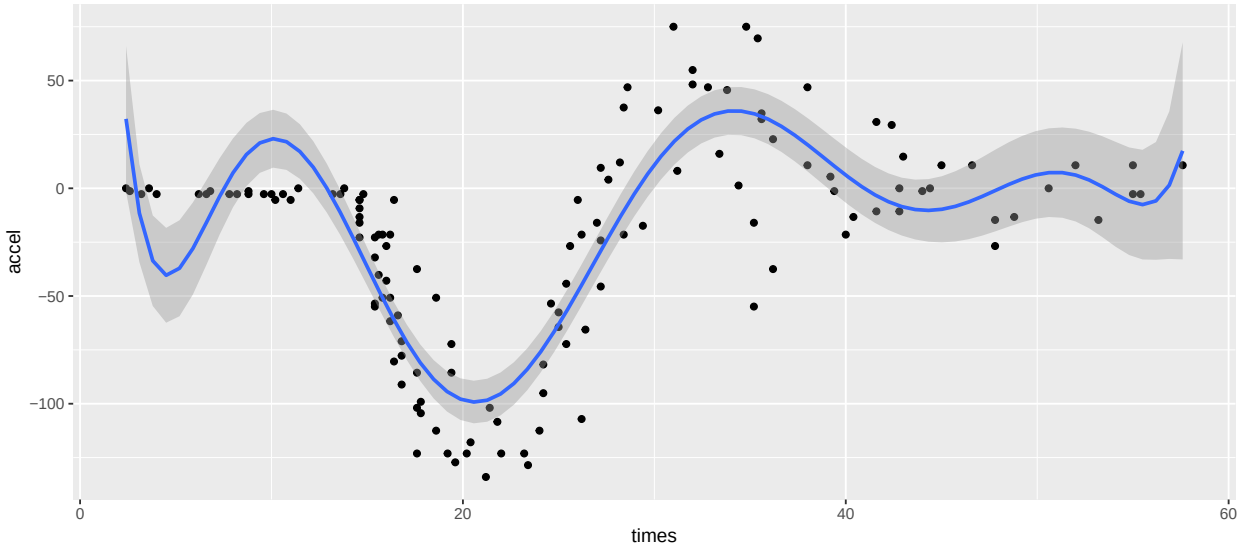
c)

```
deltas = matrix(ncol = 3, nrow = 10)
for (p in seq(10)) {
  glm =  glm(accel~poly(times, p, raw=TRUE), data=df)
  cv = cv.glm(data = df, glm = glm)
  deltas[p,] = c(p, cv$delta)
  ddf = data.frame(round(deltas, 0))
  names(ddf) = c("degree", "delta_raw", "delta_adj")
}
ddf
#>    degree delta_raw delta_adj
#> 1       1      2162      2162
#> 2       2      2057      2057
#> 3       3      1633      1633
#> 4       4      1666      1666
#> 5       5      1245      1245
#> 6       6      1162      1162
#> 7       7      1411      1408
#> 8       8       812       811
#> 9       9      2000      1990
#> 10     10      1723      1714
```

Wie wir sehen erhalten wir als bestes Modell eine polynomiale Regression von Grad 8. Zur veranschaulichung plotten wir dieses:

```
gg = ggplot(
  data = df,
  mapping = aes(
    x = times,
    y = accel
  )
)
gg = gg + geom_point()
```

```
gg + geom_smooth(
  method = 'glm',
  formula = y~poly(x, 8, raw=TRUE)
)
```



d)

Das Polynom ist von Grad 8 und besitzt somit 9 freie Parameter.

e)

```
p2 = glm(accel~poly(times, 2, raw=TRUE), data=df)
p3 = glm(accel~poly(times, 3, raw=TRUE), data=df)
p8 = glm(accel~poly(times, 8, raw=TRUE), data=df)
aic2 = round(AIC(p2), 0)
aic3 = round(AIC(p3), 0)
aic8 = round(AIC(p8), 0)
```

Wir erhalten als AIC Werte für die Modelle:

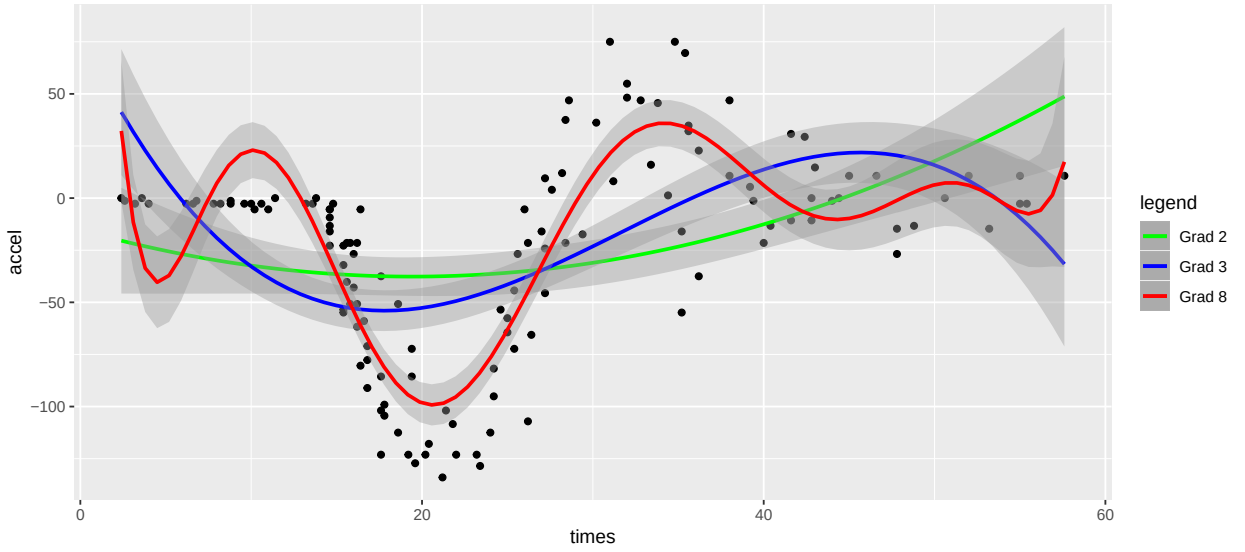
p	AIC
2	1395
3	1365
8	1262

Auch hier würden wir anhand des AIC das Modell mit Grad 8 also mit 9 Parametern wählen.

f)

```
gg = ggplot(
  data = df,
```

```
  mapping = aes(
    x = times,
    y = accel
  )
)
gg = gg + geom_point()
gg = gg + geom_smooth(
  method = 'glm',
  formula = y~poly(x, 2, raw=TRUE),
  mapping = aes(
    color = 'Grad 2'
  )
)
gg = gg + geom_smooth(
  method = 'glm',
  formula = y~poly(x, 3, raw=TRUE),
  mapping = aes(
    color = 'Grad 3'
  )
)
gg = gg + geom_smooth(
  method = 'glm',
  formula = y~poly(x, 8, raw=TRUE),
  mapping = aes(
    color = 'Grad 8'
  )
)
gg = gg + scale_colour_manual(name="legend", values=c("green", "blue", "red"))
gg
```



- Grad 2: Die Anpassung wirkt in gar keinem Bereich wirklich gut.
- Grad 3: Die Anpassung folgt besser dem verlauf, wirkt aber auch nicht wirklich gut.
- Grad 8: Von 13 bis 53 Millisekunden wirkt die Anpassung sehr gut. An den Rändern stößt am auf größere Abweichungen, die durch den hohen Interpolationsgrad zu erklären sind.