Robuste Statistik

Musterlösung zu Blatt 14

Das Regressionsmodell

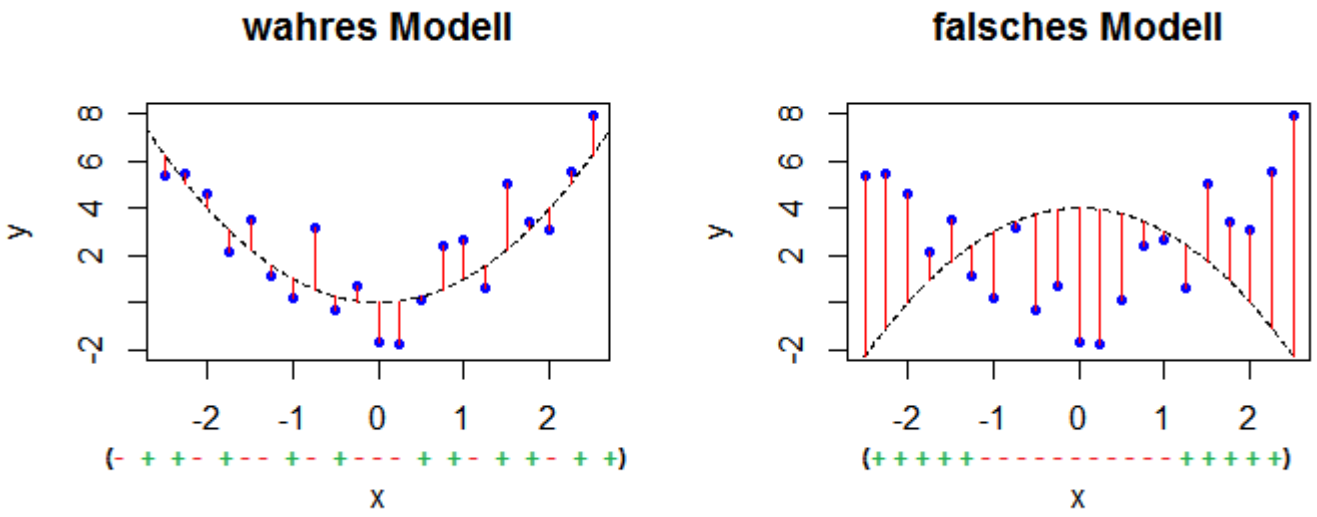
$$y_n = g(x_n, \beta) + e_n, n \in \{1, \ldots, N\}$$

beschreibt Realisationen $y_1, \ldots, y_N$ der Zufallsvariablen $Y_1, \ldots Y_N$. Dabei finden wir die stochastische Komponente im Regressionsmodell bei den Realisationen $e_1, \ldots, e_N$ der Fehler $E_1, \ldots, E_N$. Damit Maßzahlen beruhend auf Vorzeichenwechseln sinnvoll sind, müssen wir Annahmen an die Wahrscheinlichkeiten des Auftretens der Vorzeichen treffen. In vielen Modellen finden wir folgende Annahme wieder:

$$P(E_n > 0) = P(E_n < 0) = \frac{1}{2}, n \in \{1, \ldots, N\}, \tag{A}$$

d.h. mit gleicher Wahrscheinlichkeit können die Fehler positives oder negatives Vorzeichen haben. P-fast sicher wird keine 0 realisiert. Denken wir an stetigen Verteilungsklassen, so ist diese Annahme auch keine große Einschränkung. Wir können uns z.B. normalverteilte, t-verteilte, doppelexponentialverteilte, gleichverteilte, ... Fehler vorstellen, sodass der Lokationsparameter gewählt wird, dass der Median 0 ist. Ferner fordern wir, dass die Fehler stochastisch unabhängig voneinander sind (B). (Anders als im Skript dargestellt, brauchen wir keine identisch verteilten Fehler.)
Für jeden R-dimensionalen Parameter $\beta \in \mathbb{R}^R$ beschreibt $g(\bullet, \beta) : \mathbb{R} \to \mathbb{R}$ eine Regressionsfunktion abhängig von den Regressoren. Die Realisationen von Regressoren $x_1, \ldots, x_N$ der Zufallsvariablen $X_1, \ldots, X_N$ können geplant sein und damit konstanten (deterministischen) Zufallsvariablen entsprechen oder auch zufällig sein. Die Güte der Vorzeichen-Tiefe hängt von der Anordnung der Regressoren ab. Man sollte vor der Auswertung der Daten via Vorzeichen-Tiefe ein Kriterium festlegen, nach dem die Regressoren geordnet werden. Ein einfacher Ansatz ist z.B. die kanonische Ordnungsrelation auf $\mathbb{R}$ als Ordnung für $X_1 < \ldots < X_N$ zu verwenden. Je nach sachologische Fragestellung oder im mehrdimensionalen Fall können alternative Ordnungen verwendet werden.

a) Bevor die Implementierung der Vorzeichen-Tiefen dargestellt wird, soll motiviert werden, warum unter den Annahmen (A) und (B) Vorzeichen-Wechsel für Modellgüte sprechen. Angenommen, das wahre Modell beruht auf einer Parabelgleichung. Passen wir das wahre Modell an, können wir folgende Vorzeichenstruktur beobachten (siehe Abbildung links):



Unter dem wahren Modell reduzieren sich die Residuen zu den Fehlern $e_1, \ldots, e_N$. Von den Fehlern wissen wir nach Annahme (A), dass die Wahrscheinlichkeit für das Auftreten eines positiven und negativen Vorzeichen gleich ist. Wegen der Unabhängigkeitsannahme (B) erwarten wir insbesondere, dass keine langen Blöcke von gleichen Vorzeichen auftreten sollten. Daher sollte unter den Annahmen (A) und (B) ein gemischtes und wechselhaftes Verhalten der Vorzeichen beobachtet werden.

Wählen wir ein falsches Modell, z.B. mit umgedrehter Parabelöffnung (siehe Abbildung rechts), so werden wir in vielen Fällen ein anderes Verhalten der Vorzeichen beobachten. Wir haben drei Blöcke mit gleichen Vorzeichen und somit wenige Vorzeichenwechsel. Unter dem wahren Modell sollten die beobachteten Residuen den Modellfehlern $e_1, \ldots, e_N$ entsprechen. Das Verhalten spricht aber gegen Annahme (A) oder (B), weswegen wir davon ausgehen, dass das Modell unpassend ist.

Ein Implementierung in `R` kann z.B. durch den Befehl `combn()` umgesetzt werden. Dabei werden alle $\binom{N}{K}$ geordnete K-Tupel erzeugt, von denen dann das Vorzeichenwechsel-Kriterium abgeprüft werden kann.

```
signDepth <- function(res, K){
  N <- length(res)
  comb <- combn(N, K)
  fac <- rep(c(-1,1), length.out=K)
  count <- sum(apply(comb, 2, function(x){all(res[x]*fac > 0) + all(res[x]*fac < 0)}))
  d_K <-  count/choose(N,K)
  return(d_K)
}
```

Pro's und Con's dieser Implementierung

+ Implementierung für *jedes* K in einer Funktion;
+ Vektorisierte Variante ermöglicht *Parallelisierung*;
- Speicherverbrauch sehr hoch wegen der Funktion `combn` $\rightarrow$ Funktion streikt bei großem N;
- Laufzeit beträgt $\Theta(N^K)$, also für große N langsam.

Vor allem das erste Gegenargument, die Erzeugung großer Vektoren, ist problematisch, weswegen man tatsächlich von einer vektorisierten Implementierung in diesem Fall abrät und wir auf eine Implementierung via K Schleifen umsteigen. Um Rechenzeit zu gewinnen, verwenden wir das Paket `Rcpp`, das ermöglicht, C++-Implementierung in `R` einzubinden, wodurch die Schleifen deutlich schneller werden. Exemplarisch geben wir hier die Implementierung für $K = 4$ an.

```
library(Rcpp)
cppFunction('double get4Depth(NumericVector resSigns) {
            double temp=0;
            int N=resSigns.size();
            for(int i=0;i<N-3;i++){
                for(int j=i+1;j<N-2;j++){
                  for(int k=j+1;k<N-1;k++){
                    for(int l=k+1;l<N;l++){
```

```
                if((resSigns[i]<0 & resSigns[j]>0 & resSigns[k]<0 & resSigns[l]>0)
                | (resSigns[i]>0 & resSigns[j]<0 & resSigns[k]>0 & resSigns[l]<0))
              temp+= 1;
              }
            }
          }
        }
        return (temp*24/((N-1)*(N-2)*(N-3)));
        }')
```

Pro's und Con's dieser Implementierung

+ Algorithmus auch für sehr große N anwendbar, da der Algorithmus nur einen konstanten zusätzlichen Speicherbedarf braucht und die Schleifen in C++ viel schneller laufen;
- Keine Implementierung für *jedes* K in einer Funktion;
- *Parallelisierung* nicht unmittelbar möglich;
- Laufzeit beträgt $\Theta(N^K)$, also für große N schneller Anstieg der Laufzeit.

Bemerkung: Man kann die vorliegende Implementierung mit einigen Programmiertricks (durch Hilfsfunktionen) für allgemeines, eingegebenes K erweitern, sodass das erste Gegenargument eigentlich entkräftet werden kann. Allerdings ist so eine Implementierung auf den ersten Blick nicht offensichtlich.

(Kleine) Analyse der Laufzeiten
Für die (kleine) Analyse der Laufzeiten wird für $N \in \{20, 100, 500\}$ Realisierungen die Laufzeit in Sekunden von zehn wiederholten Berechnungen beider Implementierungen untersucht. Dazu nutzen wir die Funktion `microbenchmark()` aus dem gleichnamigen Paket. (Die Ergebnisse sind nicht reproduzierbar, da die Rechenleistung abhängig vom verwendeten Rechner ist. Die qualitativen Unterschiede werden aber auch auf anderen Rechnern beobachtbar sein.)

```
library(microbenchmark)

#### fuer N = 20 ####

microbenchmark(signDepth(rep(1,20), 4), times = 10L, unit = "s")
#                    expr       min        lq       mean    median        uq       max neval
#  signDepth(rep(1, 20), 4) 0.0111231 0.0111938 0.01167796 0.0113031 0.0121427 0.0127466    10

microbenchmark(get4Depth(rep(1,20)), times = 10L, unit = "s")
#                  expr      min       lq      mean   median       uq      max neval
# get4Depth(rep(1, 20)) 1.16e-05 1.17e-05 1.258e-05 1.18e-05 1.19e-05 1.86e-05    10
```

```
#### fuer N = 100 ####

microbenchmark(signDepth(rep(1,100), 4), times = 10L, unit = "s")
# expr     min       lq    mean   median      uq      max neval
# signDepth(rep(1, 100), 4) 10.8285 11.62465 11.7046 11.67378 11.9777 12.09197    10

microbenchmark(get4Depth(rep(1,100)), times = 10L, unit = "s")
#        expr       min        lq       mean    median        uq       max neval
# get4Depth(rep(1, 100)) 0.0054493 0.0054535 0.00560137 0.0054846 0.0056362 0.0063023    10

#### fuer N = 500 ####
signDepth(rep(1,500), 4)
# Error in matrix(r, nrow = len.r, ncol = count) :
#  invalid 'ncol' value (too large or NA)

microbenchmark(get4Depth(rep(1,500)), times = 10L, unit = "s")
#                 expr      min       lq     mean   median       uq      max neval
#  get4Depth(rep(1, 500)) 3.213704 3.215294 3.227242 3.223911 3.232642 3.260294    10
```

Wir können irgendwann gar nicht mehr die Laufzeiten beider Methoden vergleichen, weil die erste einen zu hohen Speicherbedarf hat!

Bemerkung: Für $K \in \{2, 3, 4, 5\}$ gibt es Möglichkeiten, die Vorzeichen-Tiefe in linearer Zeit zu implementieren. Diese Erkenntnisse beruhen auf mathematischen Ergebnissen, auf die wir in der robusten Statistik nicht mehr eingehen können. Im Abschlussprojekt werden diese Implementierungen zur Verfügung gestellt.

b) Das wahre Modell lautet

$$y_n = 3 - 2x_n + \frac{1}{2}x_n^2 + e_n, n \in \{1, \ldots, N\},$$

wobei $e_1, \ldots, e_n$ Realisationen von u.i.v. $N(0, \frac{39}{20})$-Zufallsvariablen $E_1, \ldots, E_N$ sind. Es entspricht genau dem Fall $\beta^{(3)}$.

```
## Daten
fine <- seq(-5,5, 0.01)
x <- seq(-0.75,4.75, 0.5)
set.seed(122756)

## wahres Modell
y <-  3 - 2*seq(-0.75,4.75, 0.5) + 1/2 * seq(-0.75,4.75, 0.5)^2 + rnorm(12, 0, 1.9)

## Einige Manipulationen an den Daten:
y[1] <- 10.1 + sqrt(1e-5)
y[5] <- -8 + sqrt(1e-7)
y[8] <- y[8] - 0.6

data14 <- cbind(x,y)
```

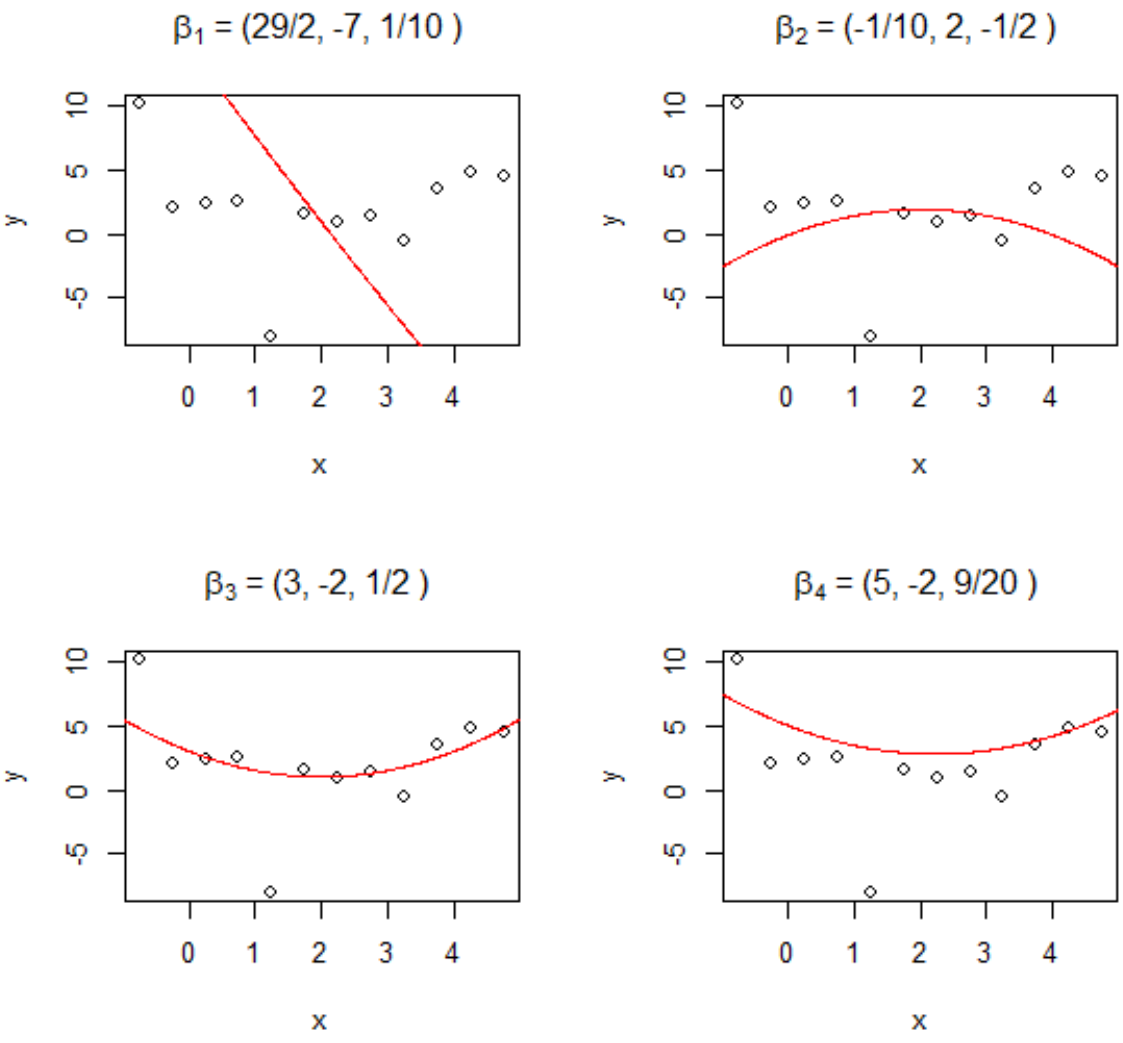
Für die Berechnung der Residuen wird der vorgeschlagene Parameter $\beta \in \mathbb{R}^3$ in die Regressionsfunktion g eingesetzt und von den Realisationen $y_1, \ldots, y_N$ abgezogen:

```
res1 <- sign(y - 14.5  + 7*seq(-0.75,4.75, 0.5) - 0.1 * seq(-0.75,4.75, 0.5)^2)
res2 <- sign(y + 1/10 - 2*seq(-0.75,4.75, 0.5) + 1/2 * seq(-0.75,4.75, 0.5)^2)
res3 <- sign(y - 3 + 2*seq(-0.75,4.75, 0.5) - 1/2 * seq(-0.75,4.75, 0.5)^2)
res4 <- sign(y - 5 + 2*seq(-0.75,4.75, 0.5) - 9/20 * seq(-0.75,4.75, 0.5)^2)
```

Für die Berechnung der Tiefen müssen wir ein Ordnungskriterium vorher festlegen und auf die Regressoren anwenden! Wir berechnen die Tiefe beruhend auf der Ordnung, die wir im Datensatz erhalten haben. Diese entspricht hier genau der kanonischen Ordnung auf $\mathbb{R}$. Daher bleibt der Datensatz unverändert und wir `res1`, ..., `res4` in die implementierten Funktionen für $K = 2, 3, 4$ einsetzen.

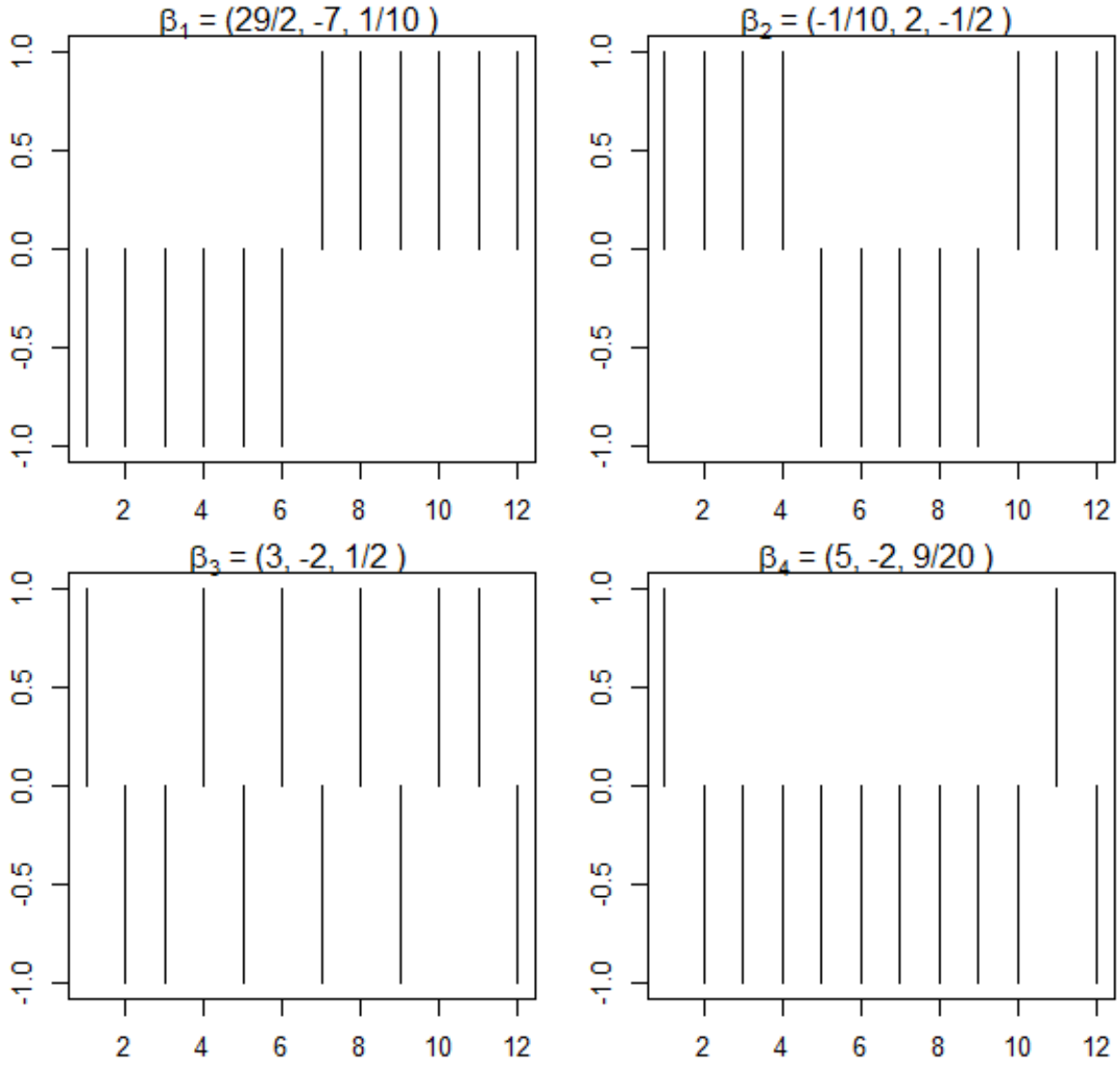
	$\beta^{(1)}$	$\beta^{(2)}$	$\beta^{(3)}$	$\beta^{(4)}$
Zweier-Tiefe	0.55	0.53	0.55	0.30
Dreier-Tiefe	0	0.27	0.32	0.08
Vierer-Tiefe	0	0	0.20	0.02

Unterstützend zur Tabelle werden unten die Daten und das vorgeschlagene Modell $\beta^{(j)}, j = 1, \ldots, 4$ dargestellt:



Das erste Modelle soll eine Halbtrennung der Daten darstellen. Das zweite Modell stellt eine falsche Orientierung der Parabel durch eine Öffnung nach unten dar. Das dritte Modell ist korrekt und das vierte Modell entspricht nahezu dem dritten Modell, bis auf eine Lageverschiebung nach oben. Wir beobachten, dass bei allen Tiefen das dritte (und wahre) Modell, eine sehr hohe Tiefe aufweist.

c) Ziel dieser Aufgabe ist das Verständnis, welchen Einfluss K bei verschiedenen Modellalternativen hat. Dazu eine Darstellung der Vorzeichen der Residuen für jedes Modell:



Das erste Modell besitzt nur einen Vorzeichenwechsel, wenn ein Residuum vom ersten Block und ein Residuum vom zweiten Block betrachtet wird. Da für eine K-Vorzeichen-Tiefe die Anzahl von $K-1$ Vorzeichenwechsel gezählt werden, kann nur die Tiefe nur für $K = 2$ größer 0 sein. Da beide Blöcke gleich groß sind, ist die Zweier-Tiefe hier auch maximal.

Das zweite Modell kann zwei Vorzeichenwechsel generieren. Daher sind sowohl für $K = 2$ als auch $K = 3$ die Tiefen positiv. Es fällt auch auf, dass die Dreier-Tiefe gar nicht so gering im Gegensatz zur Tiefe des wahren Modells mit Parameter $\beta^{(3)}$ ist..

Beim dritten (und wahren) Modell erkennen wir viele Vorzeichenwechsel.

Beim vierten Modell ergeben sich durch die Lokationsverschiebung sehr einseitige Vorzeichenstrukturen. Alle Tiefen reagieren hier sehr stark und bewerten dieses Modell sehr schlecht.

Fazit: Je nach Anzahl der Vorzeichenwechsel, die zwischen wahren und alternativen Modellen generiert werden

können, sollte man den Parameter K wählen. Bei quadratischer Regression ist der Fall $K = 3$ noch nicht empfehlenswert, da zwei Vorzeichenwechsel generiert werden können, siehe Modell 2.

Erst wenn der Stichprobenumfang sehr groß wird, können auch unpassendere K gut performen. Dies wird z.B. in dem Paper von Leckey et al. (2018) *Generalized sign tests based on sign depth* untersucht.