

Booklets

Computational Statistics / Computerintensive Methoden

24. Mai 2025

Name: Safouan Er-Ryfy

Inhaltsverzeichnis

1. Verbreitung von Krankheiten	4
A1. SIR Modell	4
a. Der Verlauf der Verbreitung simulieren	4
b. Hygienevorschriften	5
c. Kontakseinschränkung	5
d. Kombination aus HygienemaSSnahmen und Kontaktbeschränkungen	7
e. Zeit variieren	8
A2. SIS Modell	9
a. SIR und SIS vergleichen	9
b. Mögliche Erweiterung: Virusvarianten	10
c. Weitere Herausforderungen: Impfungen	10
2. Monte Carlo Integration	11
A1. Monte Carlo Approximation von Kreiszahl	11
A2. Monte Carlo Integration	13
a. n gleichverteilte Zufallszahlen	13
b. Die Verteilung der Schätzer	14
A3. Berechnen einer Fläche	15
a. Graphische Darstellung der Fläche A	15
b. Näherungsweise der Flächeninhalt	16
3. Simulation von Zufallszahlen	17
A1. Inversions- und Box-Müller-Methode	17
a. Exponentialverteilung und Poissonverteilung	17
b. Normalverteilung	18
A2. Zufallszahlengeneratoren	19
a. Random Decimal Fraction von Random.org	19
b. Mittquadratverfahren	20
c. Kreativer Generator	21
d. Mersenne-Twister und Super-Duper	22
e. Vergleich der Generatoren	22
4. Cross Validation	23
A1. Validation Set	23
a. Aufteilung in Trainings- und Testdaten	23
A2. Leave One Out Cross Validation	26
a. LOOCV	26
b. LOOCV automatisiert aus boot-Packet	27
A3. K-Folds Cross-Validation	27
5. Bootstrap	29
A1. Zentralen Grenzwertsatzes	29
a. Simulation der Daten	29

b. Bootstrap	29
c. Verteilung der Simulation	29
d. Vergleich von Means mit t-Test	31
e. Interpretieren	31
6. Shrinkage	32
Daten	32
A1. Ridge-Regression	32
a. Lineare Regression	32
b. Ridge-Regression und ggf. Koeffizientenschätzer	33
c. Plot der Koeffizienten	34
d. Cross Validation ggf. MSE Schätzer	34
e. Optimaler Schätzer für Lambda	36
A2. Lasso Regression	36
a. Lasso-Verfahren ggf. die Koeffizientenschätzer	36
b. Koeffizienten vs Lambda	37
c. Cross Validation per Hand	37
d. Optmiales Lambda	38
e. Ridge vs Lasso	38

1. Verbreitung von Krankheiten

A1. SIR Modell

Aufgabestellung:

Wir verwenden SIR (Susceptible, Infectious, Recover) Modell als *stochastic individual contact model (ICM)*.

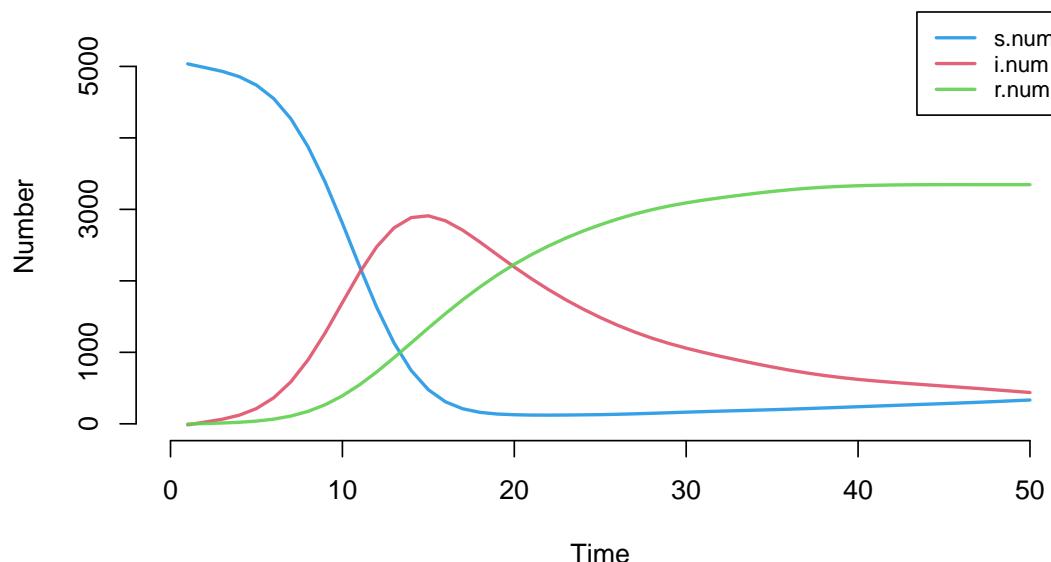
a. Der Verlauf der Verbreitung simulieren

Simulieren Sie den Verlauf der Verbreitung in 50 Tagen und stellen Sie die Anzahl von infizierbaren, infizierten und immunisierten Personen in einer geeigneten Graphik dar.

```
param <- param.icm(inf.prob = 0.1,
                     act.rate = 10,
                     rec.rate = 1/14,
                     a.rate = 1/100,
                     ds.rate = 1/90,
                     di.rate = 2/90,
                     dr.rate = 1/90)

init <- init.icm(s.num = 5000, i.num = 10, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 50)

model1 <- icm(param, init, control)
plot(model1)
```



b. Hygienevorschriften

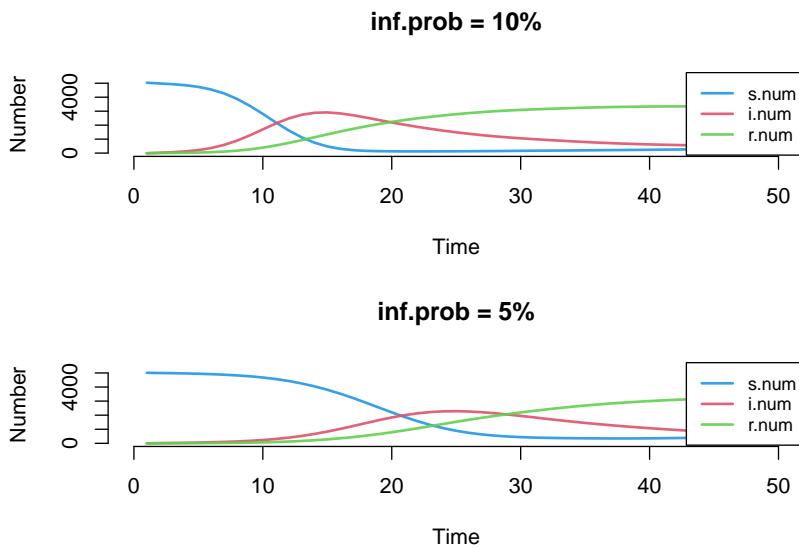
Gehen Sie davon aus, dass zum Zeitpunkt $t_H = 0$ Hygienevorschriften erlassen wurden (z.B. Händewaschen, Mundschutz etc.). Dies beeinflusst die Ansteckungswahrscheinlichkeit (*inf.prob*) und senkt sie auf 5%. Simulieren Sie die neue Situation und vergleichen Sie diese mit Ihren Ergebnissen aus a).

```
# Die Infektionswahrscheinlichkeit (inf.prob) wird von 0.1 auf 0.05 reduziert.
param <- param.icm(inf.prob = 0.05,
                     act.rate = 10,
                     rec.rate = 1/14,
                     a.rate = 1/90,
                     ds.rate = 1/90,
                     di.rate = 2/90,
                     dr.rate = 1/90)

init <- init.icm(s.num = 5000, i.num = 10, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 50)

model2 <- icm(param, init, control)

par(mfrow = c(2, 1), mar = c(4, 4, 4, 4))
plot(model1, main = "inf.prob = 10%")
plot(model2, main = "inf.prob = 5%")
```



Eine höhere Infektionswahrscheinlichkeit (*inf.prob* = 10%) führt zu einem schnelleren und stärkeren Anstieg der Infektionen. Bei niedriger Wahrscheinlichkeit (5%) verläuft die Ausbreitung deutlich flacher und langsamer.

c. Kontakseinschränkung

Gehen Sie nun davon aus, dass die Population keine Hygienemaßnahmen ergreift, aber zum Zeitpunkt $t_K = 0$ den Kontakt einschränkt. D.h. hier wird die Begegnungsrate gesenkt (auf 5 bzw. 2). Vergleichen Sie Ihre Ergebnisse mit a) und b) und interpretieren diese.

```

param1 <- param.icm(inf.prob = 0.1,
                      act.rate = 5,
                      rec.rate = 1/14,
                      a.rate = 1/90,
                      ds.rate = 1/90,
                      di.rate = 2/90,
                      dr.rate = 1/90)

param2 <- param.icm(inf.prob = 0.1,
                      act.rate = 2,
                      rec.rate = 1/14,
                      a.rate = 1/90,
                      ds.rate = 1/90,
                      di.rate = 2/90,
                      dr.rate = 1/90)

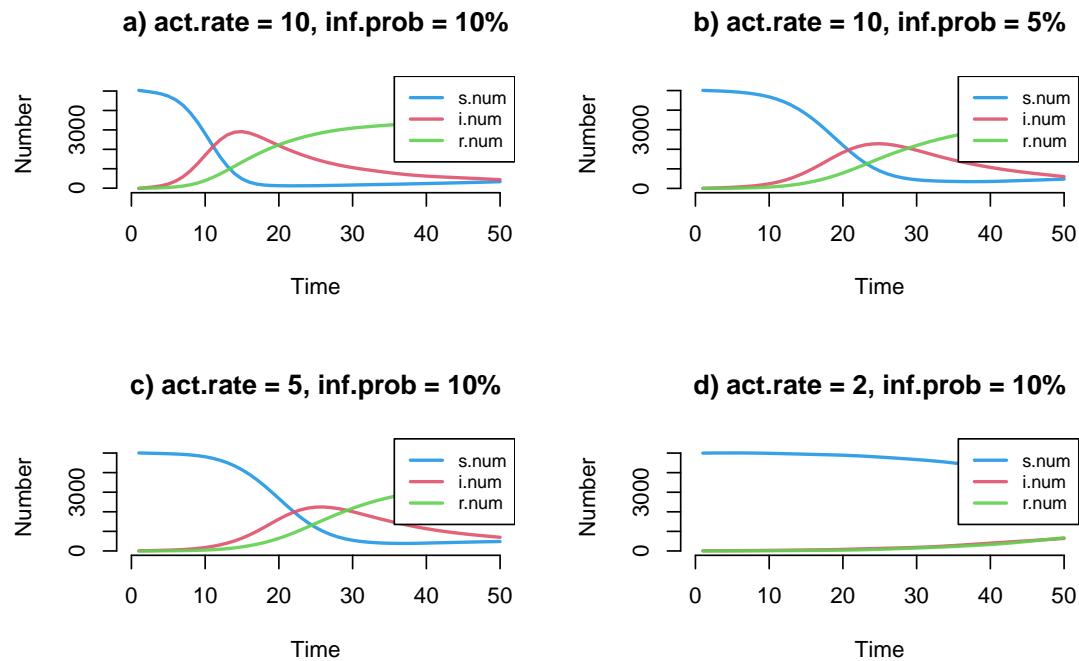
init <- init.icm(s.num = 5000, i.num = 10, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 50)

model_c1 <- icm(param1, init, control)
model_c2 <- icm(param2, init, control)

par(mfrow = c(2, 2))

plot(model1 , main = "a) act.rate = 10, inf.prob = 10%")
plot(model2 , main = "b) act.rate = 10, inf.prob = 5%")
plot(model_c1, main = "c) act.rate = 5, inf.prob = 10%")
plot(model_c2, main = "d) act.rate = 2, inf.prob = 10%")

```



- Hygienemaßnahmen und Kontaktbeschränkungen wirken sich jeweils auf die Ansteckungs-

wahrscheinlichkeit (`inf.prob`) und die Begegnungsrate (`act.rate`) aus. Die Abbildungen b) und c) zeigen: Eine Halbierung der `act.rate` hat einen ähnlichen Effekt wie eine Halbierung der `inf.prob`. Wird `act.rate` sogar auf 2 gesenkt (d)), flacht der Verlauf der Infektion deutlich ab.

d. Kombination aus HygienemaSSnahmen und Kontaktbeschränkungen

Kombinieren Sie die MaSSnahmen aus b) und c) und vergleichen auch diesen Effekt mit Ihren vorherigen Befunden.

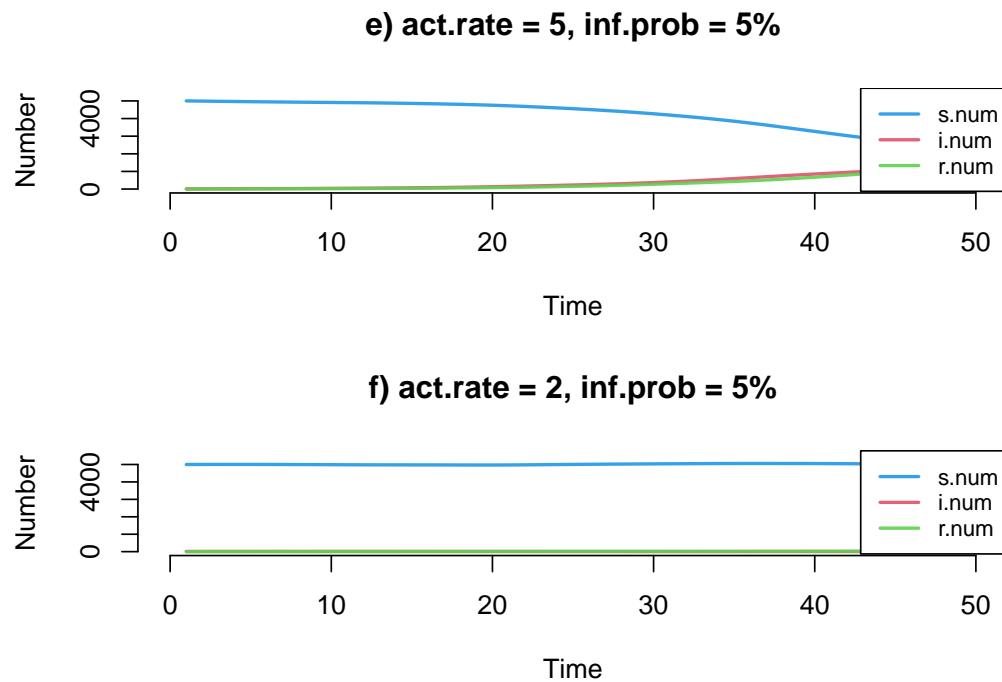
```
param3 <- param.icm(inf.prob = 0.05,
                      act.rate = 5,
                      rec.rate = 1/14,
                      a.rate = 1/90,
                      ds.rate = 1/90,
                      di.rate = 2/90,
                      dr.rate = 1/90)

param4 <- param.icm(inf.prob = 0.05,
                      act.rate = 2,
                      rec.rate = 1/14,
                      a.rate = 1/90,
                      ds.rate = 1/90,
                      di.rate = 2/90,
                      dr.rate = 1/90)

init <- init.icm(s.num = 5000, i.num = 10, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 50)

model_d3 <- icm(param3, init, control)
model_d4 <- icm(param4, init, control)

par(mfrow = c(2, 1), mar = c(4, 4, 4, 4))
plot(model_d3, main = "e") act.rate = 5, inf.prob = 5%
plot(model_d4, main = "f") act.rate = 2, inf.prob = 5%
```



- Die Kombination aus HygienemaSSnahmen und Kontaktbeschränkungen wirkt besonders effektiv – gemeinsam flachen sie die Infektionskurve deutlich stärker ab.

e. Zeit variieren

Variieren Sie die Zeitpunkte t_H und t_K und setzen den MaSSnahmen gegebenenfalls auch ein Ende. Vergleichen Sie auch jetzt die Verläufe und interpretieren diese.

- Ab dem Zeitpunkt $t_H = 10$ wurde die Ansteckungswahrscheinlichkeit (`inf.prob`) von 10% auf 5% gesenkt. Ab $t_K = 20$ kam zusätzlich eine Reduktion der Kontaktfrequenz (`act.rate`) von 10 auf 2 hinzu. Beide MaSSnahmen zusammen zeigen eine deutlich stärkere Wirkung.

A2. SIS Modell

a. SIR und SIS vergleichen

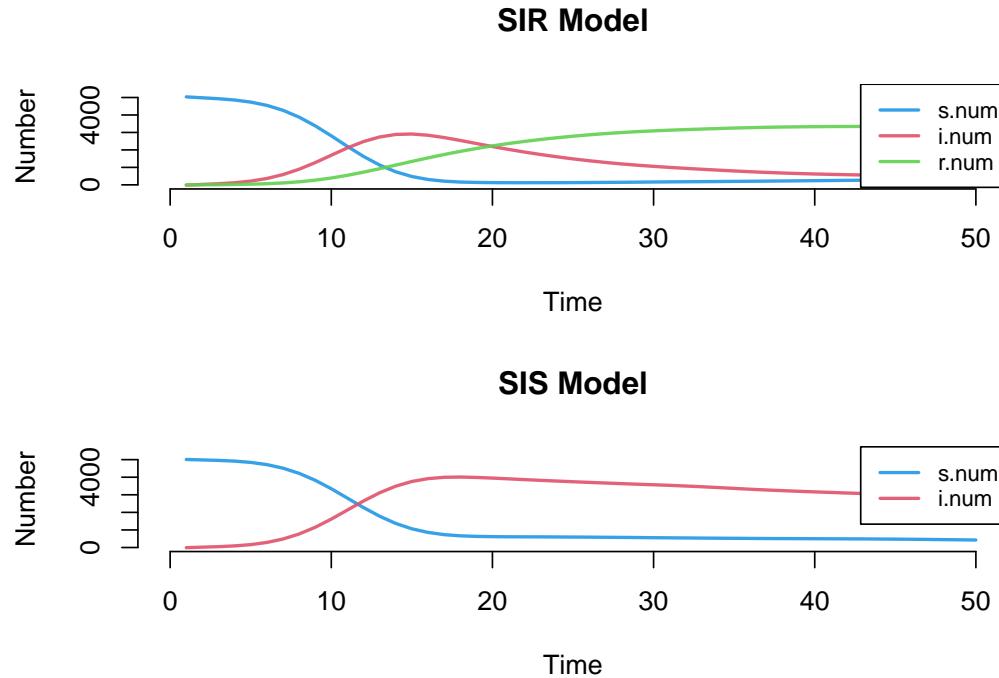
Inzwischen weiß man, dass sich Menschen nach einer gewissen Zeit erneut infizieren können.

- Was bedeutet das für unser Modell? Es gibt keine dauerhafte Immunität nach einer Genesung. Die Gruppe der Genesenen“ verliert damit an Bedeutung – stattdessen kehren Personen nach ihrer Infektion wieder in die Gruppe der Anfälligen zurück. In diesem Szenario erreicht die Anzahl der Infizierten ein stabiles Gleichgewicht, bei dem sich Neuinfektionen und Genesungen die Waage halten.
- Was wäre hier besser? Ein SIS-Modell ist für diesen Fall besser geeignet als das klassische SIR-Modell, da es genau diesen Rückfluss von Genesenen zu Anfälligen abbildet.
- Setzen Sie das neue Modell (auch als stochastic individual contact model (ICM)) in einer Simulation um, ’spielen’ mit den entsprechenden Parametern, stellen die Ergebnisse in geeigneten Graphiken dar und interpretieren Ihre Befunde.

```
param <- param.icm(inf.prob = 0.1,
                     act.rate = 10,
                     rec.rate = 1/14,
                     a.rate = 1/90,
                     ds.rate = 1/90,
                     di.rate = 2/90,
                     dr.rate = 1/90)

init <- init.icm(s.num = 5000, i.num = 10, r.num = 0)
control <- control.icm(type = "SIS", nsteps = 50)

par(mfrow = c(2, 1), mar = c(4, 4, 4, 4))
sis_model <- icm(param, init, control)
plot(model1, main="SIR Model")
plot(sis_model, main="SIS Model")
```



b. Mögliche Erweiterung: Virusvarianten

Im Verlauf der Pandemie sind viele verschiedene Virusvarianten aufgetreten.

- Was bedeuten diese Varianten für die Modellierung? Jede Variante hat eigene Eigenschaften wie Ansteckungsrate oder Krankheitsverlauf. Daher müsste man für jede Variante eigene Parameter definieren.
- Wie könnte man das modellieren – und wo liegen die Schwierigkeiten? Man könnte mehrere Varianten im Modell abbilden, z.B. mit separaten Zuständen oder Parametern.
- Probleme dabei:
 - Eine Person kann sich mit mehreren Varianten anstecken (Ko-Infektion).
 - Es ist schwierig, realistische Übertragungsraten für jede Variante zu bestimmen.
 - Die Modellierung wird deutlich komplexer, weil mehr Parameter und Zustände nötig sind.

c. Weitere Herausforderungen: Impfungen

- Wie kann man Impfungen berücksichtigen? Die Infizierbaren (S) könnten in zwei Gruppen unterteilt werden: Geimpfte und Ungeimpfte – mit unterschiedlichen Ansteckungsraten.
- Welche Parameter beeinflussen Impfungen? Besonders relevant sind `inf.prob` (Ansteckungswahrscheinlichkeit) und `rec.rate` (Erholungsrate)
- Sind diese Einflüsse konstant? Nein, sie können variieren, z.B. je nach Alter, Immunsystem, Vorerkrankungen oder Zeitpunkt der Impfung.

2. Monte Carlo Integration

A1. Monte Carlo Approximation von Kreiszahl

In dieser Übung soll die Kreiszahl approximiert werden.

Erzeuge hierzu n Tupel (x_i, y_i) , $i = 1, \dots, n$ mit

$$x_i, y_i \sim U([1, 1])$$

und definiere daraus die Variable

$$z_i = 4 \cdot \mathbf{1}(x_i^2 + y_i^2 \leq 1).$$

Hierbei ist $\mathbf{1}$ die Indikatorfunktion, die 1 liefert, wenn die Bedingung erfüllt ist, und 0 sonst.

Approximiert wird durch

$$= \frac{1}{n} \sum_{i=1}^n z_i.$$

- Erzeuge eine Tabelle, in der für $n = 10^k$, $k \in \{1, \dots, 8\}$ jeweils die Approximation und der absolute Fehler $| - |$ dargestellt werden.
- Stimmt die Genauigkeit mit der Tschebyscheff-Schätzung aus dem Video überein?

In dieser Aufgabe schätzen wir die Kreiszahl durch zufällig gezogene Punkte innerhalb des Quadrats $[1, 1]^2$.

Die Wahrscheinlichkeit, dass ein Punkt innerhalb des Einheitskreises liegt, entspricht dem Verhältnis der Flächen:

$$P(x^2 + y^2 \leq 1) = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

Da $z_i = 4 \cdot \mathbf{1}(x_i^2 + y_i^2 \leq 1)$, ist dies eine Bernoulli-Kette mit Erwartungswert:

$$E[z_i] = 4 \cdot \frac{\pi}{4} =$$

Somit ist der Stichprobenmittelwert

$$= \frac{1}{n} \sum_{i=1}^n z_i$$

eine Schätzung von .

```
set.seed(123) # Seed setzen für Reproduzierbarkeit
k = 8
alpha <- 0.05 # 95% Sicherheit

df <- data.frame(
  k = 1:k,
  n = 10^(1:k)
)

df$mean <- sapply(df$n, function(n) {
```

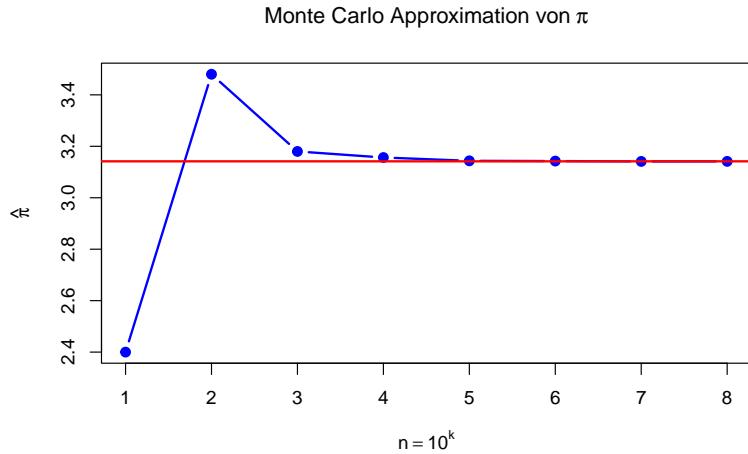
```

x <- runif(n, -1, 1) # n gleichverteilte Zufallszahlen zwischen -1 und 1 erzeugen
y <- runif(n, -1, 1)
mean(4 * (x^2 + y^2 <= 1))
})

df$var <- sapply(df$n, function(n) {
  x <- runif(n, -1, 1)
  y <- runif(n, -1, 1)
  var(4 * (x^2 + y^2 <= 1))
})

df$error <- sqrt(df$var / (alpha * df$n)) # Fehlerabschätzung basierend auf Tschebyscheff-Ungleichung

# Plot
plot(df$k, df$mean, type = "b", lwd = 2, pch = 19, col = "blue",
      xlab = expression(n == 10^k),
      ylab = expression(hat(pi)),
      main = expression("Monte Carlo Approximation von " ~ pi))
abline(h = pi, col = "red", lwd = 2)
    
```



```

df
##   k     n      mean      var      error
## 1 1 1e+01 2.400000 1.600000 1.7888543820
## 2 2 1e+02 3.480000 3.258182 0.8072399666
## 3 3 1e+03 3.180000 2.619644 0.2288948948
## 4 4 1e+04 3.156400 2.698845 0.0734689764
## 5 5 1e+05 3.143560 2.687353 0.0231834103
## 6 6 1e+06 3.142612 2.698049 0.0073458133
## 7 7 1e+07 3.141214 2.698435 0.0023231165
## 8 8 1e+08 3.141456 2.696273 0.0007343396
    
```

- Interpretation: Die Genauigkeit der Monte Carlo-Approximation von π steigt mit wachsender Stichprobengröße n , während die Varianz der Schätzungen entsprechend sinkt (Gesetz der grossen Zahlen).

A2. Monte Carlo Integration

In dieser Übung soll der Wert 1.96 als 0.975 *Quantil* der Standardnormalverteilung mit Hilfe von Monte Carlo Integration überprüft werden.

a. n gleichverteilte Zufallszahlen

Erzeuge hierzu n gleichverteilte Zufallszahlen auf dem Intervall [1,96, 1,96] und bestimme

$$2 \leq 1,96 \leq (x),$$

wobei die Dichte der Standardnormalverteilung ist, definiert durch

$$(x) = \frac{1}{2} \exp\left(-\frac{x^2}{2}\right).$$

Stelle analog zur Aufgabe A1 für verschiedene Werte von n die Approximation und ihre Genauigkeit in einer Tabelle dar. Verwende hierfür die gleichen Werte für n wie in Aufgabe A1, also $n = 10^k$ für $k \in \{1, , 8\}$.

Hinweis: Der gesuchte Integralwert ist 0,95!

```
set.seed(123) # Seed setzen für Reproduzierbarkeit
k = 8
alpha <- 0.05
Q <- 1.96 # 0.975-Quantil der Standardnormalverteilung

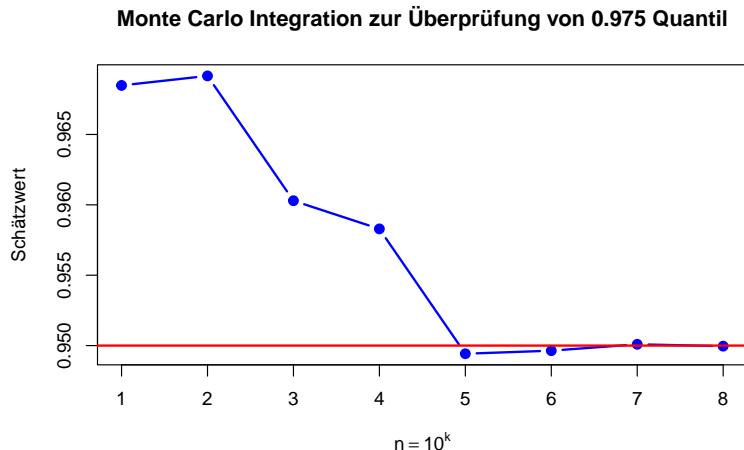
df <- data.frame(
  k = 1:k,
  n = 10^(1:k)
)

df$mean <- sapply(df$n, function(n) {
  x <- runif(n, -Q, Q) # n gleichverteilte Zufallsvariablen zwischen -Q und Q
  mean(2 * Q * dnorm(x))
})

df$var <- sapply(df$n, function(n) {
  x <- runif(n, -Q, Q)
  var(2 * Q * dnorm(x))
})

df$error <- sqrt(df$var / (alpha * df$n)) # Fehlerabschätzung basierend auf Tschebyscheff-Ungleichung

# Plot
plot(df$k, df$mean, type = "b", lwd = 2, pch = 19, col = "blue",
      xlab = expression(n == 10^k),
      ylab = "Schätzwert",
      main = "Monte Carlo Integration zur Überprüfung von 0.975 Quantil")
abline(h = 0.95, col = "red", lwd = 2)
```



```
df
##   k      n      mean      var      error
## 1 1 1e+01 0.9684838 0.2404018 0.6934000096
## 2 2 1e+02 0.9691602 0.1752135 0.1871969795
## 3 3 1e+03 0.9602923 0.1940901 0.0623041169
## 4 4 1e+04 0.9582903 0.1983122 0.0199154330
## 5 5 1e+05 0.9494218 0.1965286 0.0062694272
## 6 6 1e+06 0.9496388 0.1971466 0.0019856818
## 7 7 1e+07 0.9500912 0.1971818 0.0006279837
## 8 8 1e+08 0.9499596 0.1971159 0.0001985527
```

- Interpretation: Mit zunehmender Stichprobengröße n wird die Monte Carlo Schätzung des 0.975 Quantils der Standardnormalverteilung zunehmend genauer und stabilisiert sich sehr nah am theoretischen Wert von 0.95 (Gesetz der großen Zahlen).

b. Die Verteilung der Schätzer

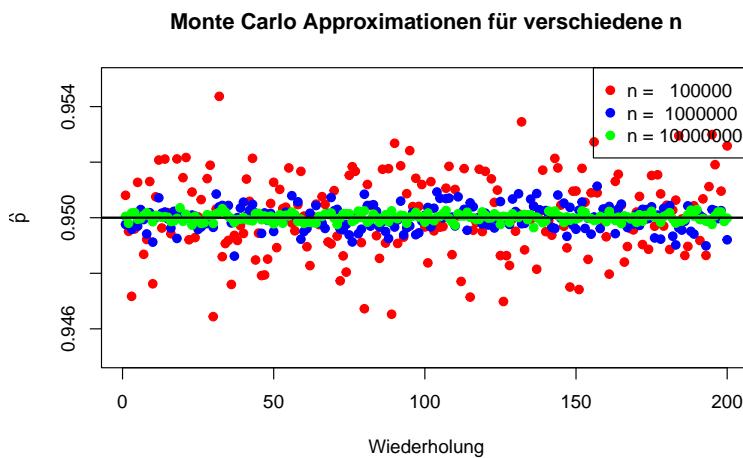
Führe für die Werte $n = 10^5$, $n = 10^6$ und $n = 10^7$ jeweils 200 Approximationen durch und stelle die Verteilung der Schätzer für $p = 0.95$ in einer gemeinsamen Abbildung dar. Interpretiere das Ergebnis.

```
set.seed(123) # Seed setzen für Reproduzierbarkeit

N <- 200 # Anzahl der Wiederholungen
n_values <- c(10^5, 10^6, 10^7)
Q <- 1.96
y_lim <- c(0.945, 0.955)
farben <- c("red", "blue", "green")

results <- list()
for (n in n_values) {
  z_mean <- replicate(N, {
    x <- runif(n, -Q, Q) # n gleichverteilte Zufallsvariablen zwischen -Q und Q
    mean(2 * Q * dnorm(x))
  })
  results[[as.character(n)]] <- z_mean
}
```

```
# Plot
plot(1:N, results[[1]], col = farben[1], pch = 19, ylim = y_lim,
      xlab = "Wiederholung", ylab = expression(hat(p)),
      main = "Monte Carlo Approximationen für verschiedene n")
for (i in 2:length(n_values)) {
  points(1:N, results[[i]], col = farben[i], pch = 19)
}
abline(h = 0.95, col = "black", lwd = 2)
legend('topright', legend = paste("n =", format(n_values, scientific = FALSE)),
       col = farben, pch = 19)
```



- Interpretation: Mit zunehmender Stichprobengröße n verringert sich die Varianz der Schätzungen signifikant. Bei $n = 10^7$ (grün) ist die Approximation extrem präzise und praktisch perfekt auf 0,95 zentriert. Die grafische Darstellung bestätigt eindrucksvoll die Konvergenzeigenschaften der Monte Carlo Integration und das Gesetz der grossen Zahlen.

A3. Berechnen einer Fläche

In dieser Übung soll eine Fläche mithilfe von Monte-Carlo-Integration berechnet werden. Als umgebendes Rechteck wählen wir $[7, 7] \times [4, 4]$, die Fläche A ist durch folgende Ungleichung definiert:

$$\frac{x^2}{49} + \frac{y^2}{9} \leq 1$$

Erzeuge $n = 10^5$ Tupel (x_i, y_i) für $i = 1, \dots, n$ mit $x_i \sim U([7, 7])$ und $y_i \sim U([4, 4])$ und bilde hieraus die Bernoulli-Variable $z_i = \mathbf{1}_A(x_i, y_i)$.

a. Graphische Darstellung der Fläche A

Stelle die Fläche A anhand der Punkte, die in A liegen, graphisch dar. Wenn die Ungleichungen richtig umgesetzt wurde, sollte Dir die Fläche bekannt vorkommen.

```
set.seed(123) # Seed setzen für Reproduzierbarkeit
n <- 10^5
x <- runif(n, min = -7, max = 7)
```

```

y <- runif(n, min = -4, max = 4)

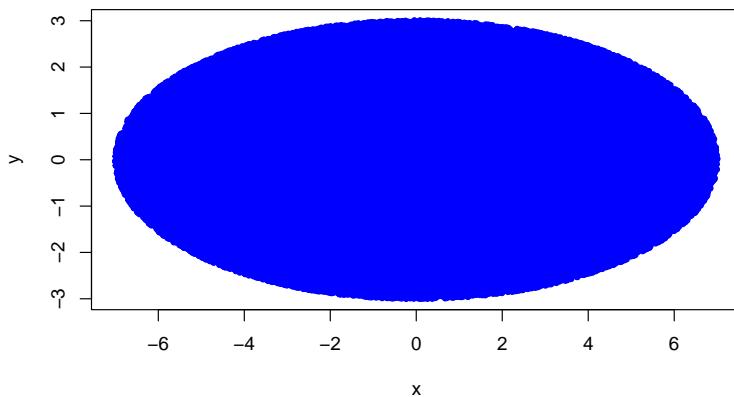
# Berechne die Funktion f(x, y) für jedes Tupel
f <- x^2 / 49 + y^2 / 9 - 1

# Bestimme die Punkte, die in A liegen
indices_A <- which(f <= 0)
x_A <- x[indices_A]
y_A <- y[indices_A]

# Plot der Punkte in A
plot(x_A, y_A, pch = 20, col = "blue",
      main = "Darstellung der Fläche A",
      xlab = "x",
      ylab = "y")

```

Darstellung der Fläche A

**b. Näherungsweise der Flächeninhalt**

Berechne näherungsweise den Flächeninhalt von A mit Hilfe der Formel aus dem Video.

$$\text{Fläche der Ellipse } (7 \times 4) \times \frac{\text{Anzahl der Punkte in } A}{n}$$

```

A <- (7*4) * (length(indices_A)/n)
print(paste("Der Flächeninhalt A beträgt näherungsweise:", A))
## [1] "Der Flächeninhalt A beträgt näherungsweise: 16.51384"

```

3. Simulation von Zufallszahlen

A1. Inversions- und Box-Müller-Methode

In dieser Aufgabe sollen aus gleichverteilten Zufallszahlen Zufallszahlen beliebiger Verteilung gewonnen werden.

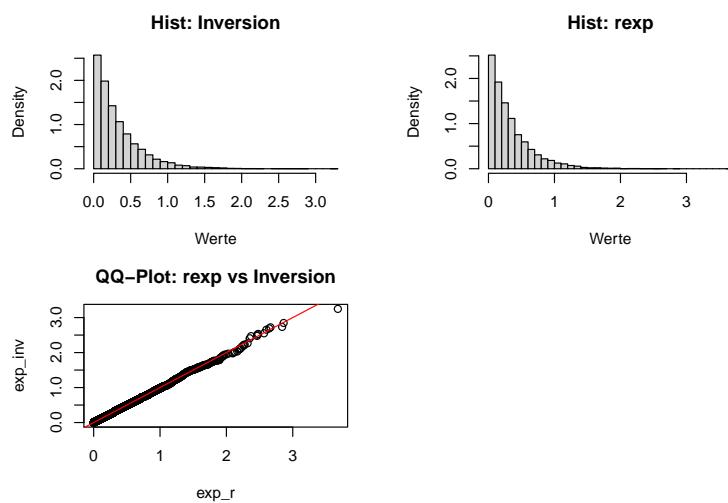
a. Exponentialverteilung und Poissonverteilung

Erzeugen Sie mit Hilfe der Inversionsmethode aus jeweils $n = 10000$ gleichverteilten Zufallszahlen zwischen 0 und 1, $n = 10000$ exponentialverteilte bzw. poissonverteilte Zufallszahlen mit Parameter = 3. Vergleichen Sie die Verteilungen mit den Verteilungen entsprechender Zufallszahlen, die direkt von R in der gewünschten Verteilung ausgegeben werden (Histogramm, QQ-Plot).

```
set.seed(123) # Seed setzen für Reproduzierbarkeit
n <- 10000
lambda <- 3
u <- runif(n, 0, 1)
exp_inv <- ((-1/lambda) * log(1 - u))
exp_r <- rexp(n, rate = lambda)

par(mfrow = c(2, 2), mar = c(4, 4, 3, 3))
hist(exp_inv, breaks = 30, probability = TRUE, main = "Hist: Inversion", xlab = "Werte")
hist(exp_r, breaks = 30, probability = TRUE, main = "Hist: rexp", xlab = "Werte")

# QQ-Plot: Vergleich von 2 Datensätzen
qqplot(exp_r, exp_inv, main = "QQ-Plot: rexp vs Inversion")
abline(0, 1, col = "red") # Diagonaöe
```



Die erste Darstellung belegt, dass die Inversionsmethode korrekt exponentiell verteilte Zufallszahlen erzeugt.

```
poisson_inver <- function(n, lambda) { # Produkt von Zufallszahlen (Knuths Algorithmus)
  x <- numeric(n)
  for (i in 1:n) {
    L <- exp(-lambda)
    p <- 1
    k <- 0
```

```

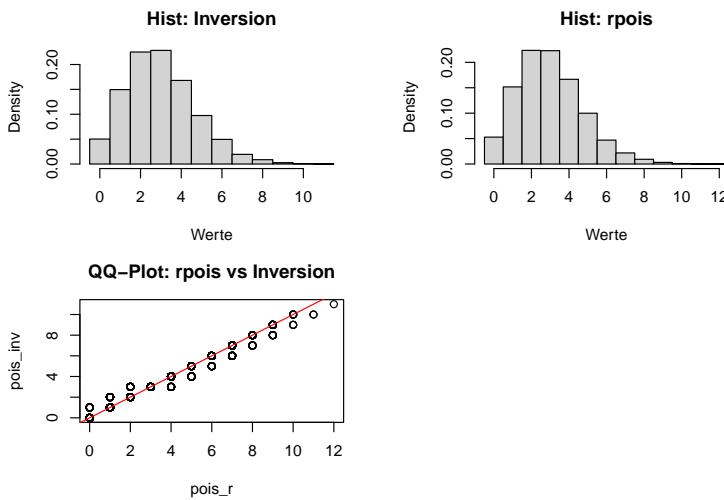
while (p > L) {
  k <- k + 1
  p <- p * runif(1)
}
x[i] <- k - 1
}
return(x)
}

pois_inv <- poisson_inver(n, lambda)
pois_r <- rpois(n, lambda)

par(mfrow = c(2, 2), mar = c(4, 4, 3, 3))
hist(pois_inv, breaks = seq(-0.5, max(pois_inv)+0.5, 1), probability = TRUE, main = "Hist: Inversion")
hist(pois_r, breaks = seq(-0.5, max(pois_r)+0.5, 1), probability = TRUE, main = "Hist: rpois", xlab = "Werte")

# QQ-Plot: Vergleich von 2 Datensätzen
qqplot(pois_r, pois_inv, main = "QQ-Plot: rpois vs Inversion")
abline(0, 1, col = "red") # Diagonale

```



Die zweite Darstellung belegt, dass die Inversionsmethode eine gute Annäherung an die echte Poissonverteilung liefert.

b. Normalverteilung

Da sich die Verteilung der Normalverteilung nicht ohne Weiteres umkehren lässt, ist die Inversionsmethode aus a) zur Erzeugung normalverteilter Zufallszahlen aus gleichverteilten Zufallszahlen ungeeignet. Verwenden Sie die Box-Müller Methode: Erzeugen Sie $n = 10000$ zweier Tupel $(u_i, v_i), i = 1, \dots, n$, aus gleichverteilten Zufallszahlen. Durch die Transformationsformel

$$z_i = \cos(2u_i) \ln(v_i)$$

lassen sich nun 10000 normalverteilte Zufallszahlen erzeugen. Vergleichen Sie auch hier die Verteilung mit den entsprechenden Zufallszahlen aus R (Histogramm, QQ-Plot).

```

u <- runif(n)
v <- runif(n)

```

```

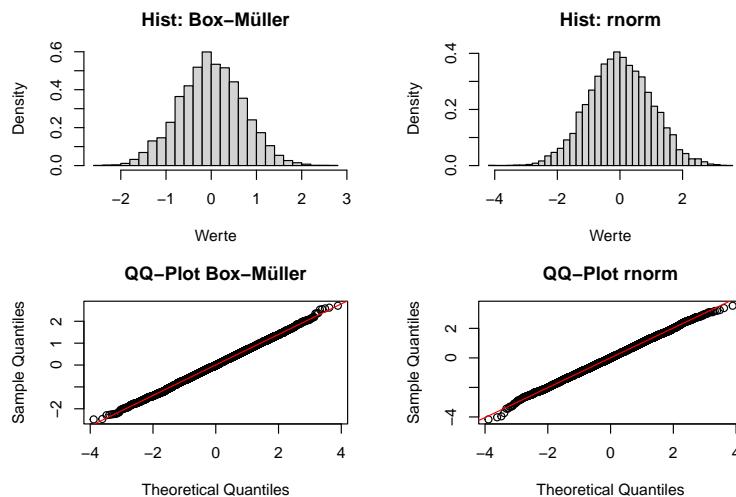
z_boxmuller <- cos(2 * pi * u) * sqrt(-log(v))
z_r <- rnorm(n)

par(mfrow = c(2, 2), mar = c(4, 4, 3, 3))
hist(z_boxmuller, breaks = 30, probability = TRUE, main = "Hist: Box-Müller", xlab = "Werte")
hist(z_r, breaks = 30, probability = TRUE, main = "Hist: rnorm", xlab = "Werte")

# QQ-Plots Normalverteilung
qqnorm(z_boxmuller, main = "QQ-Plot Box-Müller")
qqline(z_boxmuller, col = "red")

qqnorm(z_r, main = "QQ-Plot rnorm")
qqline(z_r, col = "red")

```



Die Inversionsmethode und die Box-Müller-Methode liefern Ergebnisse, die sehr gut mit den direkten R-Funktionen vergleichbar sind. Die Histogramme und QQ-Plots bestätigen die Übereinstimmung der erzeugten Verteilungen.

A2. Zufallszahlengeneratoren

In dieser Aufgabe sollen gleichverteilte Zufallszahlen auf unterschiedliche Arten erzeugt werden. Zur Überprüfung der Qualität bilden wir jeweils zweier Tupel $(z_1, z_2), (z_2, z_3), \dots, (z_n, z_n)$ und stellen diese in der zweidimensionalen Ebene dar. **Erkennbare Muster sprechen für eine schlechte Qualität der Zufallszahlen.**

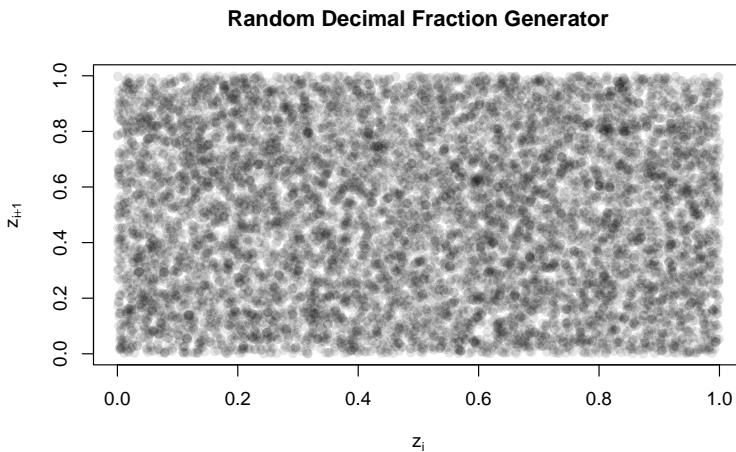
a. Random Decimal Fraction von Random.org

Lassen Sie sich auf der Seite Random.org 10000 echte gleichverteilte Zufallszahlen aus atmosphärischem Rauschen erzeugen (Random Decimal Fraction Generator)

```

# echte gleichverteilte Zufallszahlen aus atmosphärischem Rauschen
random_org <- scan("../data/random_org.txt")
plot(head(random_org, -1), tail(random_org, -1), # Paare von direkt aufeinanderfolgenden Zufallszahlen
     main = "Random Decimal Fraction Generator", xlab = expression(z[i]), ylab = expression(z[i+1]),
     pch = 19, col = rgb(0, 0, 0, 0.1))

```



b. Mittquareatverfahren

Erzeugen Sie jeweils 10000 Zufallszahlen mit dem Mittquareat Verfahren von Neumann mit $k = 8$ mittleren Ziffern.

```

mid_square <- function(n, k = 8) {
  x <- numeric(n)
  seed <- 62459921 # 8-stelliger Startwert
  for (i in 1:n) {
    square <- seed^2
    square_str <- as.character(square)

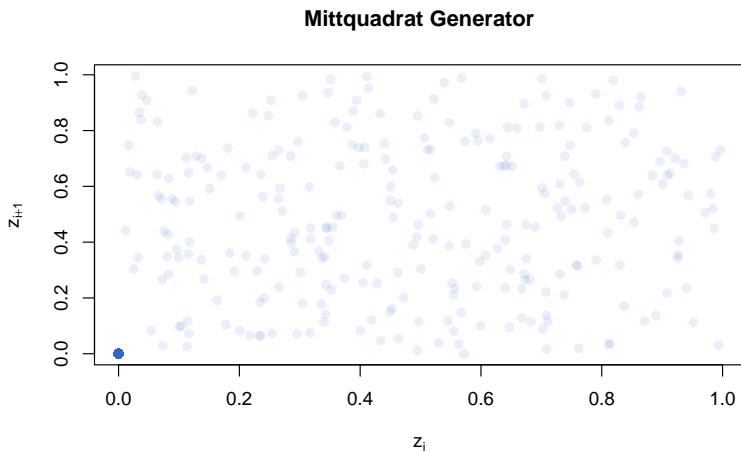
    # Auffüllen mit Nullen, falls square zu kurz ist
    while (nchar(square_str) < 2 * k) {
      square_str <- paste0("0", square_str)
    }

    # mittlere k Ziffern extrahieren
    start <- floor((nchar(square_str) - k) / 2) + 1
    mid <- substr(square_str, start, start + k - 1)

    seed <- as.numeric(mid)
    x[i] <- seed / 10^k # Skaliere auf [0, 1]
  }
  return(x)
}

msq <- mid_square(10000)
plot(head(msq, -1), tail(msq, -1),
      main = "Mittquareat Generator", xlab = expression(z[i]), ylab = expression(z[i+1]),
      pch = 19, col = c(0.2, 0.4, 0.8, 0.1))

```

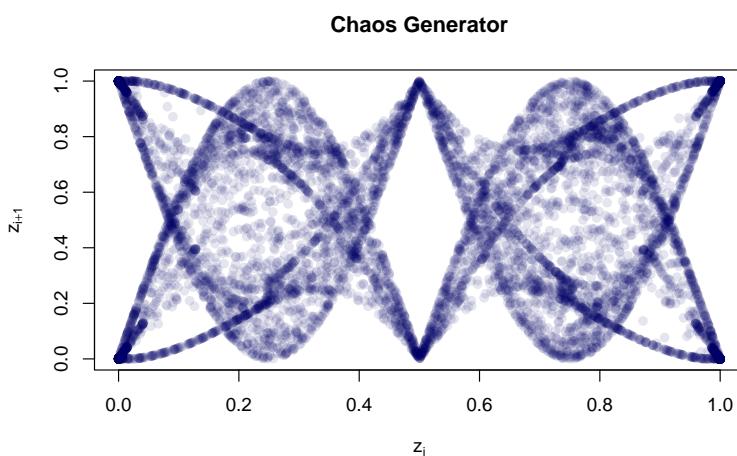


c. Kreativer Generator

Werden Sie kreativ: Kombinieren Sie die einfachen Generatoren aus der Vorlesung zu einem eigenen Zufallszahlgenerator bzw. denken Sie sich etwas völlig anderes aus - der Generator muss nicht perfekt sein! Geben Sie Ihrem Generator einen liebevollen Namen und erzeugen 10000 Zufallszahlen

```
chaos <- function(n) {
  x <- numeric(n)
  x[1] <- 0.3546 # Startwert
  for (i in 2:n) {
    x[i] <- (sin(x[i - 1] * 2 * pi) * cos(i / 20) + 2) %% 1
  }
  return(x)
}

chaos <- chaos(10000)
plot(chaos[-10000], chaos[-1],
      main = "Chaos Generator", xlab = expression(z[i]), ylab = expression(z[i+1]),
      pch = 19, col = rgb(0, 0, 0.4, 0.1))
```

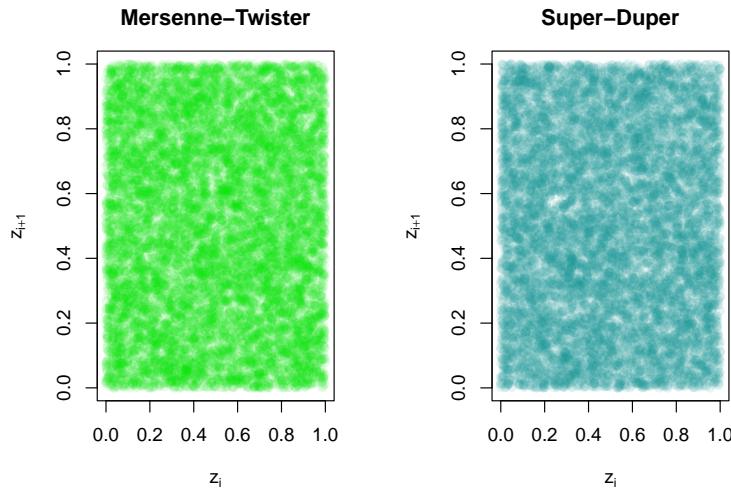


d. Mersenne-Twister und Super-Duper

Erzeugen Sie mit Hilfe des Mersenne-Twisters (Standard in R) und mit Hilfe eines beliebigen weiteren Generators in R (Funktion *RNGkind*) jeweils 10000 Zufallszahlen.

```
mersenne <- runif(10000) # Standard
RNGkind("Super-Duper")
superduper <- runif(10000)

par(mfrow = c(1, 2), mar = c(4, 4, 3, 3))
plot(head(mersenne, -1), tail(mersenne, -1),
     main = "Mersenne-Twister", xlab = expression(z[i]), ylab = expression(z[i+1]),
     pch = 19, col = rgb(0, 0.9, 0, 0.1))
plot(head(superduper, -1), tail(superduper, -1),
     main = "Super-Duper", xlab = expression(z[i]), ylab = expression(z[i+1]),
     pch = 19, col = rgb(0.1, 0.6, 0.6, 0.1))
```



e. Vergleich der Generatoren

Vergleichen Sie die erzeugten Zufallszahlen aus a)-d) mit Hilfe entsprechender Plots.

- Random Decimal Fraction: Sehr gute Gleichverteilung, keine Muster sichtbar.
- Mittsquadratverfahren: Geringe Dichte und Häufungen im Plot, Zeichen für mangelnde Zufälligkeit.
- Chaos: Erkennbares Muster, nicht als Zufallszahlengenerator geeignet.
- Mersenne-Twister: Sehr gute Gleichverteilung, keine Muster sichtbar.
- Super-Duper: Sehr gute Gleichverteilung, keine Muster sichtbar.

4. Cross Validation

A1. Validation Set

a. Aufteilung in Trainings- und Testdaten

Teile die Daten zufällig in zwei Teildatensätze vom Umfang 50 (Testdaten) und 100 (Trainingsdaten) auf. Wiederhole den Vorgang drei Mal und vergleiche die Verteilungen der 6 Variablen im jeweiligen Trainingsdatensatz mit den Verteilungen im Testdatensatz. Führe eine lineare Regression auf dem Trainingsdatensatz durch und bestimme für alle drei Aufteilungen den Test MSE und vergleiche diese. Interpretiere Deine Ergebnisse. Verwendete Funktionen: sample, predict.

```

library(dplyr)
library(Metrics)

# Load Data
load("../data/Donald.RData")
data <- Donald_1
N <- nrow(data)
target <- "Trump"

train_size <- 100
test_size <- 50

mse_values <- c()
train_dist <- list()
test_dist <- list()

formula <- as.formula(paste(target, " ~ .")) # alle 6 Variablen
for (i in 1:3) {
  # Zufällige Aufteilung der Daten
  train_idx <- sample(1:N, train_size)
  test_idx <- sample(setdiff(1:N, train_idx), test_size)

  train <- data[train_idx, ]
  test <- data[test_idx, ]

  # Verteilungen speichern
  train_dist[[i]] <- train
  test_dist[[i]] <- test

  # Lineare Regression auf dem Trainingsdatensatz
  model <- lm(formula, data = train)

  # Vorhersagen für den Testdatensatz
  pred_test <- predict(model, test)
  # Berechnung von MSE
  mse_values[i] <- mse(test[[target]], pred_test)
}

# MSE über alle Wiederholungen
print(cbind(mse_values))
##      mse_values
## [1,] 35.98927

```

a. Aufteilung in Trainings- und Testdaten

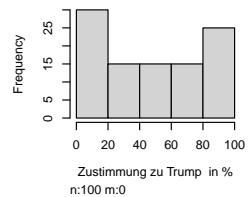
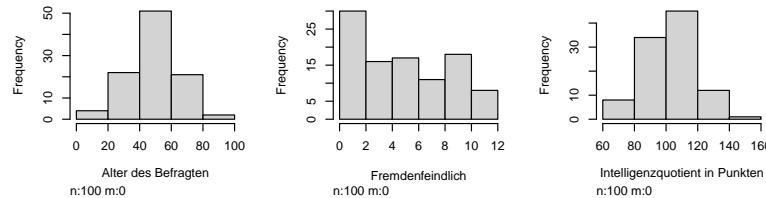
A1. VALIDATION SET

```
## [2,] 30.66994
## [3,] 35.81006
avg_mse <- mean(mse_values)

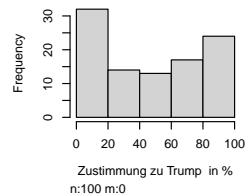
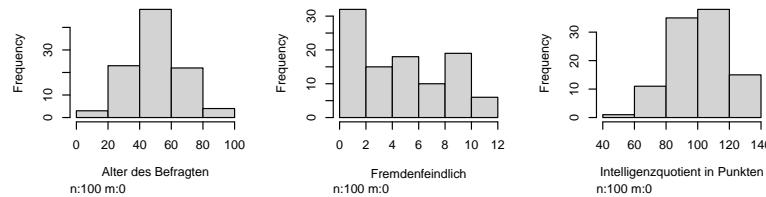
# Durchschnitt
print(cbind(avg_mse))
##      avg_mse
## [1,] 34.15642

library(Hmisc) # Macht automatisch mehrere Histogramme für alle numerischen Spalten

# Verteilung der Trainingsdatensatz
# Erster Gang
hist.data.frame(data.frame(train_dist[[1]]))
```



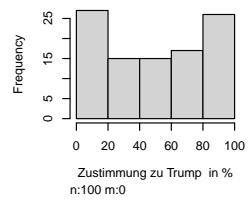
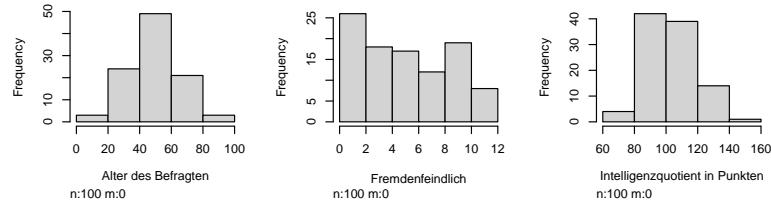
```
# Zweiter Gang
hist.data.frame(data.frame(train_dist[[2]]))
```



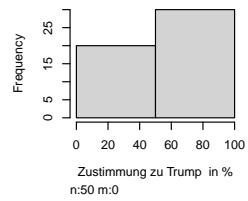
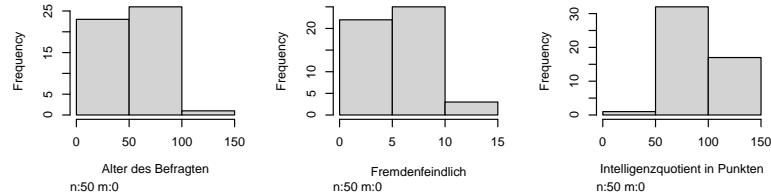
```
# Dritter Gang
hist.data.frame(data.frame(train_dist[[3]]))
```

a. Aufteilung in Trainings- und Testdaten

A1. VALIDATION SET

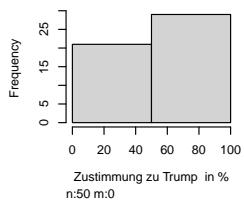
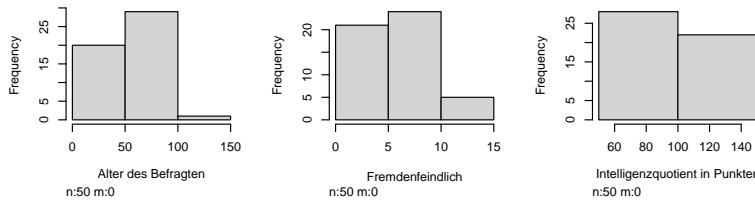


```
# Verteilung der Testdatensatz  
# Erster Gang  
hist.data.frame(data.frame(test_dist[[1]]))
```

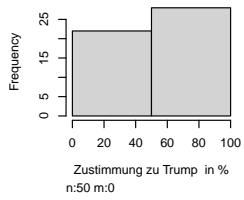
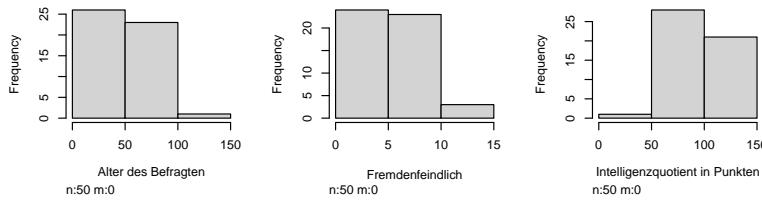


```
# Zweiter Gang  
hist.data.frame(data.frame(test_dist[[2]]))
```

A2. LEAVE ONE OUT CROSS VALIDATION



```
# Dritter Gang
hist.data.frame(data.frame(test_dist[[3]]))
```



- Die Verteilungen der sechs Variablen aus den verschiedenen zufälligen Aufteilungen sehen insgesamt ähnlich aus. Es gibt nur kleine Unterschiede, die statistisch nicht ins Gewicht fallen. Die Eigenschaften der Daten bleiben also stabil, auch wenn die Aufteilung wechselt.
- Die mittleren quadratischen Fehler (MSE) der drei Trainingsläufe liegen bei 35.99, 30.67 und 35.81. Der durchschnittliche Fehler beträgt 34.16. Diese Schwankungen zeigen, dass die Modellgüte zwar leicht variiert, insgesamt aber auf einem vergleichbaren Niveau bleibt.

A2. Leave One Out Cross Validation

a. LOOCV

Führe ein Leave-One-Out Cross Validation mit Hilfe einer `for` Schleife

```
mse_values <- c()
for (i in 1:N) {
  # Alle Daten auer der aktuellen
  train <- data[-i, ]
```

```

model <- lm(formula, data = train)
pred_train <- predict(model, newdata = data[i, ])

mse_values[i] <- mse(pred_train, as.numeric(data[i, target]))
}

avg_mse <- mean(mse_values)
# MSE für Leave-One-Out-Kreuzvalidierung
print(cbind(avg_mse))
##           avg_mse
## [1,] 38.19052

```

- Im Vergleich zum durchschnittlichen MSE aus Aufgabe 1 (34.16) liegt der LOOCV-Wert etwas höher. Das zeigt, dass das Modell bei LOOCV leicht schlechter abschneidet, was daran liegen kann, dass LOOCV empfindlich auf einzelne Ausreißer reagiert.

b. LOOCV automatisiert aus boot-Packet

Führe Leave-One-Out Cross Validation automatisiert mit Hilfe der Funktionen `glm` und `cv.glm` (Paket `boot`) durch und vergleiche das Ergebnis mit dem Ergebnis aus a).

```

#install.packages("boot")
library(boot)
res <- cv.glm(data, glmfit = glm(formula, data = data), K = N)
mse <- res$delta[1]
# MSE für automatisierte Leave-One-Out-Kreuzvalidierung
print(cbind(mse))
##           mse
## [1,] 38.19052

```

- MSE von beiden Aufgaben A2.a und A2.b sind identisch.

A3. K-Folds Cross-Validation

Führe jeweils 10 k-fache Cross Validations mit $k = 5$ und $k = 10$ mit Hilfe der Funktionen `glm` und `cv.glm` durch. Vergleiche die Ergebnisse sowohl untereinander als auch mit den Ergebnissen aus Aufgabe 1 und Aufgabe 2.

```

# K=10
res <- cv.glm(data, glmfit = glm(formula, data = data), K = 10)
mse_k10 <- res$delta[1]

# MSE für 10 k-fache Kreuzvalidierungen
print(cbind(mse_k10))
##           mse_k10
## [1,] 37.81481

# K=5
res <- cv.glm(data, glmfit = glm(formula, data = data), K = 5)
mse_k5 <- res$delta[1]

# MSE für 5 k-fache Kreuzvalidierungen
print(cbind(mse_k5))
##           mse_k5

```

```
## [1] 38.29088
```

- Die Ergebnisse aller Methoden liegen im ähnlichen Bereich (zwischen ca. 34 und 38), was zeigt, dass die Modellleistung insgesamt stabil und verlässlich ist.
- Die zufälligen Aufteilungen ergeben den niedrigsten MSE, könnten aber zu leicht optimistischen Schätzungen führen.
- LOOCV ist sehr genau, aber auch rechenintensiv und empfindlich gegenüber Ausreißern.
- 10-Fold-CV liefert ein Ergebnis, das nahe an LOOCV liegt – bei viel geringerem Rechenaufwand. Es ist deshalb eine sehr gute praktische Wahl.

5. Bootstrap

A1. Zentralen Grenzwertsatzes

Der Zentrale Grenzwertsatz besagt, dass die Verteilung der Durchschnitt einer groSSen Anzahl von unabhängigen und identisch verteilten Zufallsvariablen näherungsweise normalverteilt ist, unabhängig von der ursprünglichen Verteilung der Variablen.

In dieser Aufgabe soll die Aussage des Zentralen Grenzwertsatzes für Bootstrap-Daten überprüft werden.

a. Simulation der Daten

Simuliere 50 bzw. 1000 Datensätze mit jeweils 500 gleichverteilten Daten auf $[0, 1]$, speichere die Mittelwerte im Vektor $Means_{sim}$

```
set.seed(123)

N <- 500
datasets_50 <- replicate(50, runif(N, 0, 1))
datasets_1000 <- replicate(1000, runif(N, 0, 1))

Means_sim_50 <- apply(datasets_50, 2, mean)
Means_sim_1000 <- apply(datasets_1000, 2, mean)
```

b. Bootstrap

Simuliere einen weiteren Datensatz mit 500 gleichverteilten Daten auf $[0, 1]$ und erzeuge mit Bootstrap 50 bzw. 1000 Replikationen. Speichere die Mittelwerte der Replikationen im Vektor $Means_{boot}$

```
library(boot)

data <- runif(n, 0, 1) # 500 gleichverteilten Daten
bootstrap_mean <- function(data, indices) {
  return(mean(data[indices]))
}

# Bootstrap-Replikationen
N_boot_50 <- 50
N_boot_1000 <- 1000

boot_results_50 <- boot(data = data, statistic = bootstrap_mean, R = N_boot_50)
Means_boot_50 <- boot_results_50$t

boot_results_1000 <- boot(data = data, statistic = bootstrap_mean, R = N_boot_1000)
Means_boot_1000 <- boot_results_1000$t
```

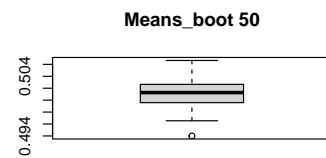
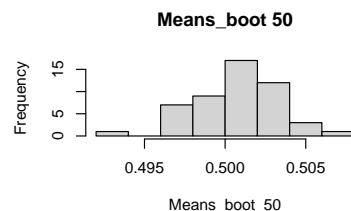
c. Verteilung der Simulation

Vergleiche die Histogramme und Boxplots von $Means_{sim}$ und $Means_{boot}$, teste jeweils mit dem Shapiro-Wilk Test auf Normalverteilung.

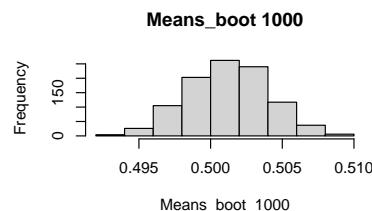
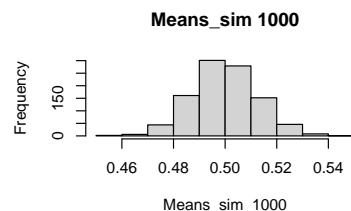
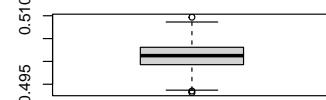
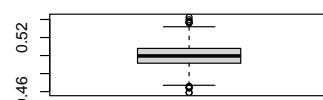
```
par(mfrow = c(2, 2))
hist(Means_sim_50, main = "Means_sim 50")
```

```
## Error in eval(expr, envir, enclos): Objekt 'Means_sim_50' nicht gefunden
hist(Means_boot_50, main = "Means_boot 50")
boxplot(Means_sim_50, main = "Means_sim 50")
## Error in eval(expr, envir, enclos): Objekt 'Means_sim_50' nicht gefunden
boxplot(Means_boot_50, main = "Means_boot 50")

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



```
hist(Means_sim_1000, main = "Means_sim 1000")
hist(Means_boot_1000, main = "Means_boot 1000")
boxplot(Means_sim_1000, main = "Means_sim 1000")
boxplot(Means_boot_1000, main = "Means_boot 1000")
```

**Means_sim 1000****Means_boot 1000**

```
# Shapiro-Wilk Test
shapiro.test(Means_sim_50)
## Error in eval(expr, envir, enclos): Objekt 'Means_sim_50' nicht gefunden
shapiro.test(Means_boot_50)
##
##  Shapiro-Wilk normality test
##
```

```
## data: Means_boot_50
## W = 0.98361, p-value = 0.7108
shapiro.test(Means_sim_1000)
##
## Shapiro-Wilk normality test
##
## data: Means_sim_1000
## W = 0.99869, p-value = 0.6836
shapiro.test(Means_boot_1000)
##
## Shapiro-Wilk normality test
##
## data: Means_boot_1000
## W = 0.99853, p-value = 0.5724
```

Die Ergebnisse der Shapiro-Wilk Tests zeigen, dass sowohl die simulierten Mittelwerte als auch die Bootstrap-Mittelwerte keine signifikante Abweichung von der Normalverteilung aufweisen. Durch die zunehmende Anzahl der Replikationen werden die Verteilungen von beiden Simulationsmethoden die Normalverteilung approximieren.

d. Vergleich von Means mit t-Test

Vergleiche die Mittelwerte von *Means_{sim}* und *Means_{boot}* mit einem t-Test.

```
t.test(Means_sim_50, Means_boot_50)
## Error in eval(expr, envir, enclos): Objekt 'Means_sim_50' nicht gefunden
t.test(Means_sim_1000, Means_boot_1000)
##
## Welch Two Sample t-test
##
## data: Means_sim_1000 and Means_boot_1000
## t = -3.3085, df = 1096.6, p-value = 0.0009683
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -0.0021343947 -0.0005452353
## sample estimates:
## mean of x mean of y
## 0.4998570 0.5011968
```

e. Interpretieren

Interpretiere und begründe Deine Beobachtungen.

- Für t-Test *Means_{sim}_50* und *Means_{boot}_50*: Der p-Wert ist größer als < 0.05. Das bedeutet, dass wir die Nullhypothese nicht verwerfen können, es gibt keinen Unterschied zwischen den Mittelwerten der beiden Gruppen.
- Für t-Test *Means_{sim}_1000* und *Means_{boot}_1000*: Der p-Wert ist auch viel kleiner als 0.05, es gibt keinen signifikanten Unterschied zwischen den beiden Daten.
- Größere Stichproben neigen dazu, Unterschiede besser zu erkennen und präzisere Schätzungen zu liefern.

6. Shrinkage

Daten

prostates Daten:

- lcalvol: TumorVolumen (logarithmiert)
- lweight: Tumorweight (log)
- age: Alter der Patienten
- lbph: Anzahl Veränderungen (log)
- gleason: Gleason Score
- pgg45: 4/5
- lpc: Organkapsel
- svi: Sammelbläschen
- lpsa: PSA (log) Frage: Scale von Variablen sind unterschiedlich -> Ergebnisse beeinflussen?

-> Unterschiedliche Skalierungen der Variablen können die Ergebnisse beeinflussen. Bei der Verwendung von Schrinkage Methoden ist es wichtig, alle Variablen auf einer vergleichbaren Skala zu liegen, da unskalierten Variablen könnte eine unfaire Gewichtung zugewiesen werden.

A1. Ridge-Regression

In diesem Übungsblatt wird der Datensatz prostate verwendet. Hier wird der Einuss verschiedener Variablen von Prostata-Krebs-Patienten auf deren PSA-Wert untersucht.

Beschreibung der Parameter des Prostate-Datensatzes:

- lcalvol: TumorVolumen (log)
- lweight: Tumorweight (log)
- age: Alter der Patienten
- lbph: Anzahl der Veränderungen (log)
- gleason: Gleason Score
- pgg45: Anteil des Tumors mit Gleason (in Prozent)
- lpc: Anteil der Organkapsel (in Prozent)
- svi: Sammelbläschen (ja/nein)
- lpsa: PSA-Wert (log)

a. Lineare Regression

In dieser Aufgabe soll die Ridge-Regression auf den Datensatz angewendet werden. Führe eine lineare Regression mit den UV lcalvol, lweight, age, lbph, svi, lpc, gleason und pgg45 und der AV lpsa durch und speichere die Koeffizienten in einem Vektor.

```
#install.packages("caret")
library(caret)

load("../data/prostate.rdata")
head(prostate)

##      lcalvol lweight age      lbph svi      lcp gleason pgg45      lpsa
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294       6      0 -0.4307829
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294       6      0 -0.1625189
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294       7     20 -0.1625189
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294       6      0 -0.1625189
## 5  0.7514161 3.432373 62 -1.386294 0 -1.386294       6      0  0.3715636
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294       6      0  0.7654678
##   train
```

```

## 1 TRUE
## 2 TRUE
## 3 TRUE
## 4 TRUE
## 5 TRUE
## 6 TRUE

UV <- c("lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason", "pgg45")
AV <- "lpsa"

# Pre-processing transformation (centering, scaling etc.)
preProcValues <- preProcess(prostate[, c(UV, AV)], method = c("center", "scale"))
scaled_data <- predict(preProcValues, prostate[, c(UV, AV)])
scaled_data$train <- prostate$train # 'train' Spalte

# Lineares Modell mit skalierten Daten
model <- lm(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45, data = scaled_data)
coeffs_scaled <- coef(model)
print("Koeffizienten des linearen Modells (skalierte Daten):")
## [1] "Koeffizienten des linearen Modells (skalierte Daten):"
print(coeffs_scaled)
## (Intercept) lcavol lweight age lbph
## 2.363183e-16 5.762193e-01 2.308529e-01 -1.370452e-01 1.215521e-01
## svi lcp gleason pgg45
## 2.731707e-01 -1.284605e-01 3.079639e-02 1.089116e-01

```

b. Ridge-Regression und ggf. Koeffizientenschätzer

Führe eine Ridge-Regression durch und vergleiche die Koeffizientenschätzer für $\lambda = 0$ und $\lambda = 10$ mit den Schätzern aus a). Interpretiere die Ergebnisse. Hinweis: Die Ridge-Regression wird mit der Funktion glmnet (Paket glmnet) mit der Option $\alpha = 0$ durchgeführt. Diese Funktion benötigt als Input die Modell Matrix der linearen Regression (Funktion model.matrix), sowie den Vektor der Werte der AV (lpsa).

```

library("glmnet")
X <- as.matrix(scaled_data[, UV]) # Matrix
y <- scaled_data[, AV] # Vektor
ridge_model_0 <- glmnet(x = X, y = y, alpha = 0, lambda = 0)
ridge_model_10 <- glmnet(x = X, y = y, alpha = 0, lambda = 10)

coeffs_0 <- coef(ridge_model_0)
coeffs_10 <- coef(ridge_model_10)
print(coeffs_0)
## 9 x 1 sparse Matrix of class "dgCMatrix"
## 
## (Intercept)    s0
## (Intercept) 7.729658e-17
## lcavol      5.761480e-01
## lweight     2.309267e-01
## age        -1.370643e-01
## lbph       1.215089e-01
## svi        2.731499e-01
## lcp        -1.284792e-01
## gleason    3.095103e-02
## pgg45      1.088483e-01

```

```
print(coeffs_10)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## (Intercept) 2.705404e-17
## lcavol      5.860717e-02
## lweight     3.519802e-02
## age         1.031090e-02
## lbph        1.436453e-02
## svi         4.349529e-02
## lcp         3.992858e-02
## gleason    2.543938e-02
## pgg45      2.939740e-02
```

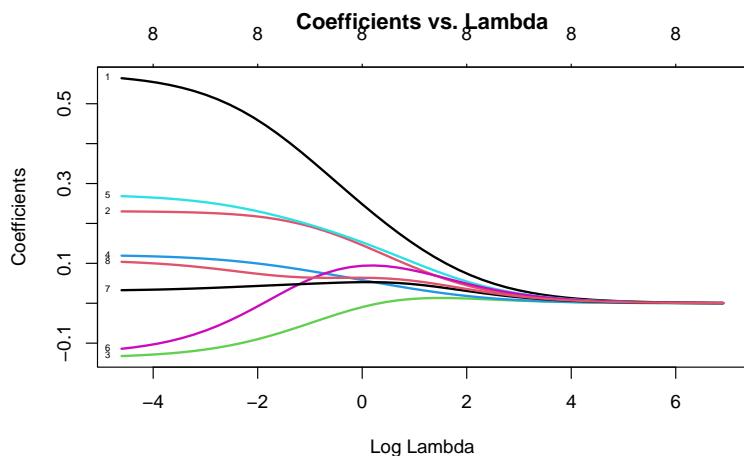
- Bei $\lambda = 0$ stimmen die Ridge-Koeffizienten fast exakt mit den Schätzern aus a) überein. Das ist zu erwarten.
- Mit $\lambda = 10$ werden die Schätzer Richtung Null „gezogen“, was die Regularisierung zeigt.

c. Plot der Koeffizienten

Erzeuge einen Plot der Koeffizienten für verschiedene λ -Werte.

```
lambda_values <- 10^seq(3, -2, by = -.1)

ridge_model <- glmnet(x = X, y = y, alpha = 0, lambda = lambda_values)
plot(ridge_model, xvar = "lambda", label=T, main = "Coefficients vs. Lambda", lw = 2)
```



- Alle Koeffizienten werden gleichzeitig geschrumpft. Für steigende λ konvergieren alle UV gleichzeitig gegen Null.

d. Cross Validation ggf. MSE Schätzer

```
# Funktion zur Berechnung des Mittleren Quadratischen Fehlers (MSE)
mse_calculate <- function(y_true, y_pred) {
  mean((y_true - y_pred)^2) # Durchschnitt
}

# Funktion zur Durchführung von k-facher Cross-Validation für Ridge-Regression
k_fold_cv <- function(X, y, lambda_values, k = 5) {
```

```

n <- nrow(X)
fold_size <- floor(n / k)           # Größe jedes Folds
mse_values <- numeric(length(lambda_values)) # Vektor zur Speicherung der MSEs für jedes Lambda

# Schleife über alle Lambda-Werte
for (lambda_idx in 1:length(lambda_values)) {
  lambda <- lambda_values[lambda_idx]      # Aktueller Lambda-Wert
  mse_folds <- numeric(k)                  # MSEs für jedes Fold

  # Schleife über die k Folds
  for (i in 1:k) {
    # Indizes für Test- und Trainingsdaten bestimmen
    test_indices <- ((i-1) * fold_size + 1):(i * fold_size) # Test-Daten-Indizes
    if (i == k) { # Letzter Fold kann größer sein, um alle Daten abzudecken
      test_indices <- ((i-1) * fold_size + 1):n
    }
    train_indices <- setdiff(1:n, test_indices) # Trainings-Daten-Indizes

    # Trainings- und Testdaten erstellen
    X_train <- X[train_indices, ]
    y_train <- y[train_indices]
    X_test <- X[test_indices, ]
    y_test <- y[test_indices]

    # Ridge-Modell trainieren mit alpha=0 (reine Ridge-Regression) und aktuellem Lambda
    model <- glmnet(X_train, y_train, alpha = 0, lambda = lambda)

    # Vorhersagen für die Testdaten
    y_pred <- predict(model, s = lambda, newx = X_test)

    # MSE für diesen Fold berechnen
    mse_folds[i] <- mse_calculate(y_test, y_pred)
  }

  # Durchschnittliche MSE über alle Folds speichern
  mse_values[lambda_idx] <- mean(mse_folds)
}

return(mse_values) # Rückgabe der MSEs für alle getesteten Lambda-Werte
}
# Beispielhafte Anwendung
lambda_values <- c(0.0, 0.09, 2)
mse_values <- k_fold_cv(X,y, lambda_values, k = 5)
cbind(mse_values)
##      mse_values
## [1,] 0.7121494
## [2,] 0.7421922
## [3,] 1.0012264

```

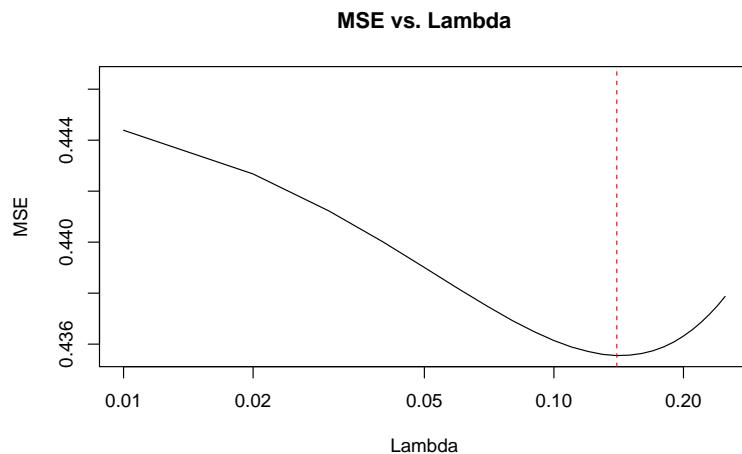
- Cross Validation mit ($k = 5$) zeigt, dass für $\lambda = 0$ (lineare Regression) hat das Modell die beste Leistung ($MSE = 0.7121494$).

e. Optimaler Schätzer für Lambda

Ermittle mithilfe der Funktion cv.glmnet den optimalen Schätzer für . Erzeuge einen Plot für den Test-MSE-Schätzer abhängig von . Vergleiche das Ergebnis mit d).

```
# Cross-validated Ridge Regression mit cv.glmnet durchführen
set.seed(1)
lambda_values <- seq(0, 0.25, by = 0.01)
cvfit <- cv.glmnet(x = X, y = y, alpha = 0, lambda = lambda_values, type.measure = c("mse"))
optimal_lambda_ridge <- cvfit$lambda.min

# Plot der Test-MSE-Schätzer abhängig von Lambda
plot(cvfit$lambda, cvfit$cvm, type = "l", log = "x", xlab = "Lambda", ylab = "MSE", main = "MSE vs.
abline(v = optimal_lambda_ridge, col = "red", lty = 2)
```



Der Unterschied in den MSE-Werten (0.71 vs. 0.43) entsteht, weil cv.glmnet eine optimierte, integrierte Kreuzvalidierung verwendet, die die beste und robustere Fold-Splits wählt. Im Vergleich ist die manuelle k_fold_cv-Funktion weniger genau, da sie einfachere Fold-Aufteilungen und keine interne Standardisierung nutzt.

A2. Lasso Regression

a. Lasso-Verfahren ggf. die Koeffizientenschätzer

```
lasso_model_0 <- glmnet(x = X, y = y, alpha = 1, lambda = 0)
lasso_model_10 <- glmnet(x = X, y = y, alpha = 1, lambda = 10)

coeffs_0 <- coef(lasso_model_0)
coeffs_10 <- coef(lasso_model_10)
print(coeffs_0)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 7.729658e-17
## lcavol      5.761480e-01
## lweight     2.309267e-01
## age        -1.370643e-01
## lbph       1.215089e-01
```

```

## svi      2.731499e-01
## lcp     -1.284792e-01
## gleason 3.095103e-02
## pgg45   1.088483e-01
print(coeffs_10)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## (Intercept) 3.469447e-17
## lcavol     0.000000e+00
## lweight    .
## age        .
## lbph       .
## svi        .
## lcp        .
## gleason   .
## pgg45     .

```

- Lasso mit groSSem = 10 eliminiert unwichtige Variablen (Feature-Selektion).
- Ridge (= 0) hingegen schrumpft alle Koeffizienten, aber setzt keine auf exakt null.

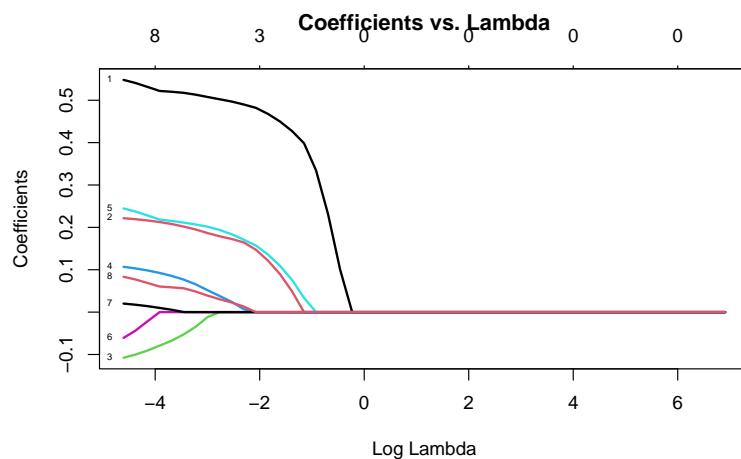
b. Koeffizienten vs Lambda

```

lambda_values <- 10^seq(3, -2, by = -.1)

lasso_model <- glmnet(x= X, y= y, alpha = 1, lambda = lambda_values)
plot(lasso_model,xvar = "lambda", label=T, main = "Coefficients vs. Lambda", lw = 2)

```



- Viele Koeffiziente wurden schnell direkt gegen Null geschrumpft im Vergleich zum Ridge-Regression.

c. Cross Validation per Hand

```

# Beispielhafte Anwendung
lambda_values <- c(0, 0.002, 1)
mse_values <- k_fold_cv(X,y, lambda_values, k = 5)
cbind(mse_values)

```

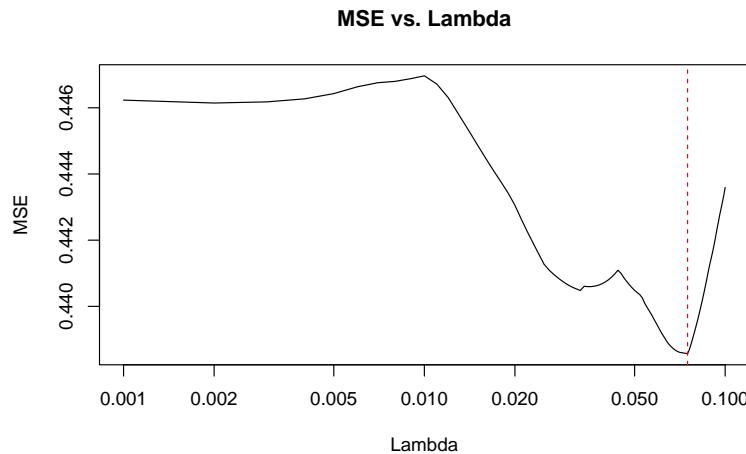
```
##      mse_values
## [1,] 0.7121494
## [2,] 0.7128475
## [3,] 0.9070115
```

- Linear-Regression haben in diesem Fall von Cross Validation per Hand die kleinste Summe der quadratischen Residuen ($\$MSE=0.7121494 \$$)

d. Optmiales Lambda

```
set.seed(1)

lambda_values <- seq(0, 0.1, by = 0.001)
cvfit <- cv.glmnet(x = X, y = y, alpha = 1, lambda = lambda_values, type.measure = c("mse"))
optimal_lambda_lasso <- cvfit$lambda.min
# Plot der Test-MSE-Schätzer abhängig von
plot(cvfit$lambda, cvfit$cvm, type = "l", log = "x", xlab = "Lambda", ylab = "MSE", main = "MSE vs.
abline(v = optimal_lambda_lasso, col = "red", lty = 2)
```



e. Ridge vs Lasso

```
# Koeffizienten des Ridge-Modells anzeigen
ridge_model_opt <- glmnet(x = X, y = y, alpha = 0, lambda = optimal_lambda_ridge)
print("Ridge Regression Koeffizienten")
## [1] "Ridge Regression Koeffizienten"
print(coef(ridge_model_opt))
## 9 x 1 sparse Matrix of class "dgCMatrix"
##          s0
## (Intercept) 5.338649e-17
## lcavol      4.560267e-01
## lweight     2.168684e-01
## age        -8.895733e-02
## lbph       9.914075e-02
## svi        2.295260e-01
## lcp       -5.275766e-03
## gleason    4.361341e-02
```

```

## pgg45      7.324929e-02

# Koeffizienten des Ridge-Modells anzeigen
lasso_model_opt <- glmnet(x = X, y = y, alpha = 1, lambda = optimal_lambda_lasso)
print("Lasso Regression Koeffizienten")
## [1] "Lasso Regression Koeffizienten"
print(coef(lasso_model_opt))
## 9 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## (Intercept) 7.471329e-18
## lcavol      4.983479e-01
## lweight     1.742249e-01
## age         .
## lbph        2.779657e-02
## svi         1.863210e-01
## lcp         .
## gleason    .
## pgg45      2.466259e-02

```

Interpretieren:

- Die Koeffizienten von *age*, *lcp* und *gleason* bei **Lasso Regression** wurden auf Null gesetzt. Das bedeutet, dass diese Variable keinen Beitrag zur Vorhersage des Modells leistet, zumindest nicht signifikant genug im Vergleich zu den anderen Prädiktoren, die im Modell verbleiben. Lasso bietet die Variablenelektion, da irrelevante oder schwach beitragende Variablen aus dem Modell entfernt werden können, was zu einem spärlichen Modell führt.
- Ridge-Regression tendiert dazu, alle Variablen beizubehalten und ihre Koeffizienten proportional zu ihrer Bedeutung für die Modellvorhersage zu schrumpfen. Je größer, desto stärker werden die Koeffizienten reduziert, aber sie bleiben immer noch größer als Null.