# Übungsblatt 2 - Lösung

## Aufgabe 2.1

Finde für die gegebene Folge $a_n$ zwei einfachere Folgen $b_n$ und $c_n$, sodass $a_n = O(b_n)$ und $a_n = o(c_n)$.

(a) $a_n = 100n^{-1} + 2n^{0.5} + 50n^{1.5}$,
(b) $a_n = 2\log(n)/n + \log(2)/n^2 + \log(1 + 1/n)$.

### Lösung

Wir erinnern uns, dass $a_n = O(b_n)$, falls $\limsup_{n\to\infty} |a_n/b_n| < \infty$, und $a_n = o(c_n)$, falls $\lim_{n\to\infty} |a_n/b_n| = 0$.

(a) Zum Beispiel $b_n = n^{1.5}$ und $c_n = n^2$, denn

$$\frac{a_n}{b_n} = \frac{100n^{-1} + 2n^{0.5} + 50n^{1.5}}{n^{1.5}} = 100n^{-2.5} + 2n^{-1} + 50 \overset{n\to\infty}{\to} 50.$$

und ähnlich zeigt man $a_n/c_n \to 0$. Optional kann man auch in O-Notation umformen:

$$a_n = O(n^{-1}) + O(n^{0.5}) + O(n^{1.5}) = O(n^{1.5}) + O(n^{1.5}) + O(n^{1.5}) = O(n^{1.5}) = O(b_n)$$

und, weil $b_n = o(c_n)$, gilt auch $a_n = O(b_n) = o(c_n)$.

(b) Zum Beispiel $b_n = \log(n)/n$ und $c_n = 1$. Es gilt

$$\frac{2\log(n)/n}{\log(n)/n} \to 2 \quad \Rightarrow 2\log(n)/n = O(b_n)$$

und

$$\frac{\log(2)/n^2}{\log(n)/n} = \frac{\log(2)}{\log(n)n} \to 0 \quad \Rightarrow \log(2)/n^2 = o(b_n).$$

Außerdem gilt $\log(1 + 1/n) \leq 1/n$ (zum Beispiel aus Mittelwertsatz) und damit

$$\frac{\log(1+1/n)}{\log(n)/n} \leq \frac{1}{\log(n)} \to 0, \quad \Rightarrow \log(1 + 1/n) = o(b_n).$$

Insgesamt ist also $a_n = O(b_n)$. Da $b_n \to 0$, gilt $b_n = o(1)$ und damit auch $a_n = O(b_n) = o(1)$.

## Aufgabe 2.2

Sei $\epsilon_n$ eine Folge und $a_n = O(\epsilon_n + \epsilon_n^2)$. Zeige, dass $a_n = O(\epsilon_n^2)$, falls $\epsilon_n \to \infty$, aber $a_n = O(\epsilon_n)$, falls $\epsilon_n \to 0$.

### Lösung

Für $\epsilon_n \to \infty$ erhalten wir

$$a_n/\epsilon_n^2 = (\epsilon_n + \epsilon_n^2)/\epsilon_n^2 = \epsilon_n^{-1} + 1 \to 1.$$

Also gilt $\epsilon_n + \epsilon_n^2 = O(\epsilon_n^2)$ und damit $a_n = O(\epsilon_n + \epsilon_n^2) = O(\epsilon_n^2)$.

Für $\epsilon_n \to 0$ gilt dagegen

$$a_n/\epsilon_n^2 = (\epsilon_n + \epsilon_n^2)/\epsilon_n^2 = \epsilon_n^{-1} + 1 \to \infty.$$

Also $a_n \neq O(\epsilon_n^2)$. Allerdings ist

$$a_n/\epsilon_n = (\epsilon_n + \epsilon_n^2)/\epsilon_n = 1 + \epsilon_n \to 1$$

und damit $a_n = O(\epsilon_n)$.

## Aufgabe 2.3

Seien $\boldsymbol{X}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \in \mathbb{R}^d$ unabhängig und identisch verteilte Zufallsvektoren mit $\mathbb{V}\mathrm{ar}(\boldsymbol{X}) = \Sigma$. Für jeden Parameter $\boldsymbol{\beta}$ gilt

$$\sigma^2(\boldsymbol{\beta}) = \mathbb{V}\mathrm{ar}(\boldsymbol{\beta}^\top \boldsymbol{X}) = \boldsymbol{\beta}^\top \Sigma \boldsymbol{\beta}.$$

Die Kovarianzmatrix $\Sigma$ lässt sich wie folgt schätzen:

$$\widehat{\Sigma} = \frac{1}{n} \sum_{i=1}^{1} (\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}})(\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}})^\top, \qquad \widehat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{X}_i.$$

Entsprechend definieren wir $\widehat{\sigma}^2(\boldsymbol{\beta}) = \boldsymbol{\beta}^\top \widehat{\Sigma} \boldsymbol{\beta}$. Aus diesen Formeln können wir den folgenden Algorithmus zur Berechnung von $\widehat{\sigma}^2(\boldsymbol{\beta})$ ableiten:

```
#' @param X Matrix mit Zeilen X_1, ..., X_n.
#' @param beta Parametervektor
```

```
sigma_2 <- function(X, beta) {
  n <- nrow(X)
  d <- ncol(X)
  Sig <- matrix(0, d, d)
  for (i in 1:n) {
    Sig <- Sig + (X[i, ] - colMeans(X)) %*% t(X[i, ] - colMeans(X))
  }
  as.numeric(t(beta) %*% Sig %*% beta / n)
}
```

(a) Gebe die Laufzeit- und Speicher-Komplexität des Algorithmus in Landau-Notation an.

**Lösung**

- Speicherplatz für `X`: $O(nd)$
- Speicherplatz für `beta`: $O(d)$
- Speicherplatz für `Sig`: $O(d^2)$

Insgesamt haben wir also $O(nd + d^2)$.

- Laufzeit für `colMeans(X)`: $O(nd + d) = O(nd)$ ($n - 1$ Additionen für jede der $d$ Spalten und $d$ Multiplikationen für die Durchschnittsbildung)
- Laufzeit für `(X[i, ] - colMeans(X)) %*% t(X[i, ] - colMeans(X))`: $O(nd + d + d^2) = O(nd + d^2)$ ($O(nd)$ Operationen für `colMeans(X)`, $2d$ Subtraktionen, $d^2$ Multiplikationen für das äußere Produkt$)
- $n$ Iterationen über `i`.

Insgesamt haben wir also $O(n(nd + d^2)) = O(nd^2 + n^2 d)$.

(b) Überprüfe die Laufzeiteigenschaften numerisch. Erzeuge dazu mehrere Datensätze mit $n \in [10, 1000]$ und $d \in [10, 1000]$ und benchmarke den Algorithmus (z.B. mit `microbenchmark::microbenchmark()`).

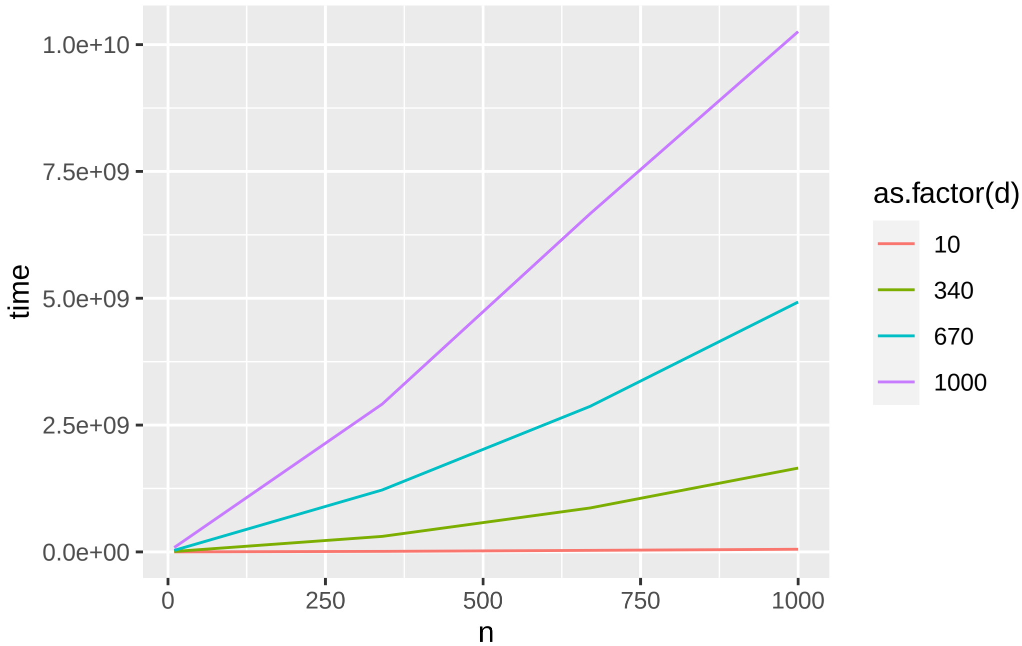
**Lösung**

```
library(microbenchmark)
library(ggplot2)
```

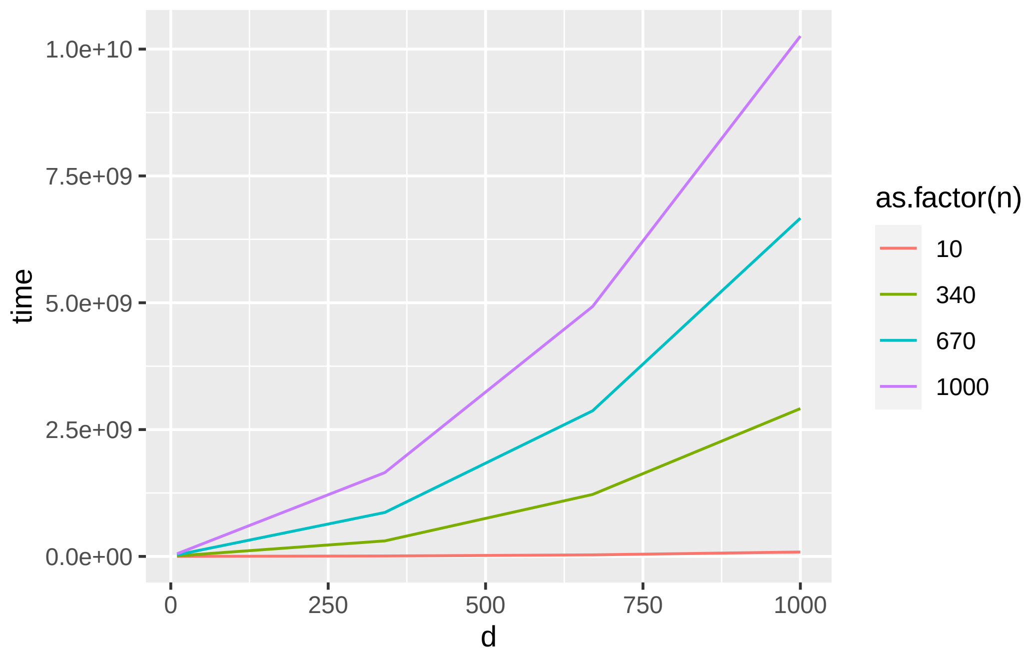
```
grid <- expand.grid(
 n = seq.int(10, 1000, length = 4),
 d = seq.int(10, 1000, length = 4)
)

times <- numeric(nrow(grid))

for (k in 1:nrow(grid)) {
 n <- grid[k, 1]
 d <- grid[k, 2]
 X <- replicate(d, rnorm(n))
 cat("n = ", n, ", d = ", d, ", time = ", sep = "")
 times[k] <- median(microbenchmark(sigma_2(X, rep(1, d)), times = 10)$time)
 cat(times[k] / 1e9, "\n")
}
```

```
n = 10, d = 10, time = 0.0002089095
n = 340, d = 10, time = 0.01028803
n = 670, d = 10, time = 0.03044275
n = 1000, d = 10, time = 0.05171079
n = 10, d = 340, time = 0.008129915
n = 340, d = 340, time = 0.3056225
n = 670, d = 340, time = 0.8662746
n = 1000, d = 340, time = 1.652354
n = 10, d = 670, time = 0.03062369
n = 340, d = 670, time = 1.221027
n = 670, d = 670, time = 2.869393
n = 1000, d = 670, time = 4.925347
n = 10, d = 1000, time = 0.08680942
n = 340, d = 1000, time = 2.913856
n = 670, d = 1000, time = 6.666899
n = 1000, d = 1000, time = 10.2561
```

```
cbind(grid, time = times) |>
 ggplot(aes(n, time, color = as.factor(d))) +
 geom_line()
```




```
cbind(grid, time = times) |>
 ggplot(aes(d, time, color = as.factor(n))) +
 geom_line()
```




(c) Finde und benchmarke einen alternativen Algorithmus mit Laufzeit- und Speicherkomplexität $O(nd)$.

**Lösung**

Die Kernideen sind:

- Der Mittelwert $\widehat{\boldsymbol{\mu}}$ muss nicht in jeder Schleifeniteration neu berechnet werden. Das bringt uns von $O(nd^2 + n^2 d)$ auf $O(nd^2 + nd) = O(nd^2)$.
- Es gilt

$$\boldsymbol{\beta}^\top \widehat{\Sigma} \boldsymbol{\beta} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\beta}^\top (\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}})(\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}})^\top \boldsymbol{\beta} = \frac{1}{n} \sum_{i=1}^{n} Z_i^2$$

mit

$$Z_i = \boldsymbol{\beta}^\top (\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}}) = \boldsymbol{\beta}^\top \boldsymbol{X}_i - \boldsymbol{\beta}^\top \widehat{\boldsymbol{\mu}}.$$

$Z_i^2$ lässt sich in $O(d)$ Operationen berechnen, der Durchschnitt $1/n \sum_{i=1}^{n} Z_i^2$ in $O(n)$.

```
#' @param X Matrix mit Zeilen X_1, ..., X_n.
#' @param beta Parametervektor
sigma_2_besser <- function(X, beta) {
  Xbeta <- X %*% beta
  mu <- mean(Xbeta)
  sig <- 0
  for (i in 1:n) {
    sig <- sig + (Xbeta[i] - mu)^2
  }
  sig / n
}
```

Die Schleife können wir auch noch beschleunigen, indem wir die R-Funktion `mean()` verwenden.

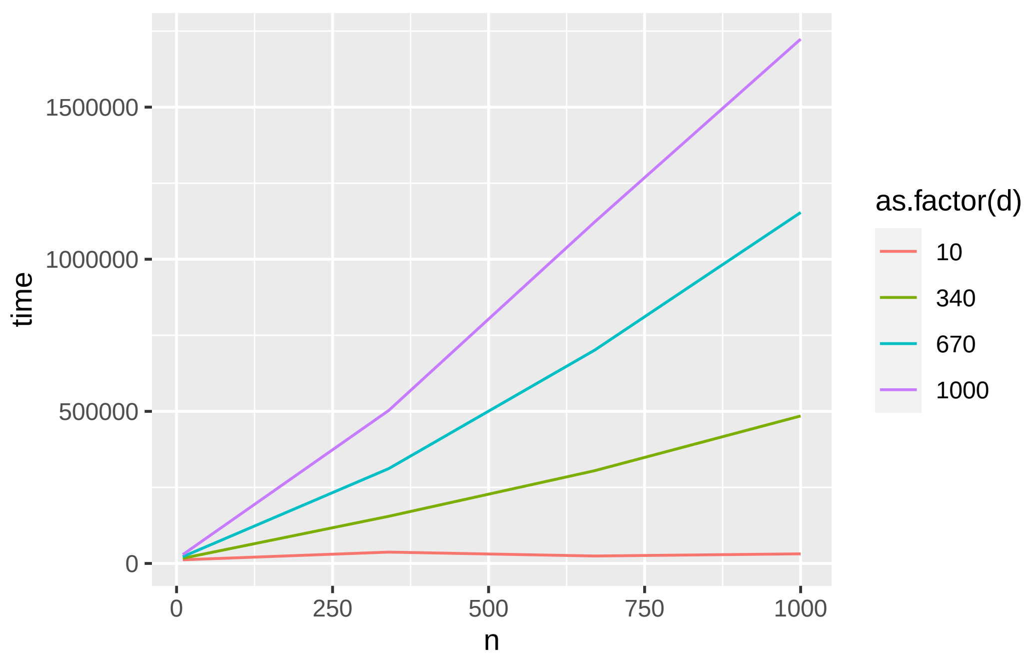
```
#' @param X Matrix mit Zeilen X_1, ..., X_n.
#' @param beta Parametervektor
sigma_2_besser <- function(X, beta) {
  Xbeta <- X %*% beta
  mu <- mean(Xbeta)
  mean((Xbeta - mu)^2)
}
```

```
grid <- expand.grid(
 n = seq.int(10, 1000, length = 4),
 d = seq.int(10, 1000, length = 4)
)

times <- numeric(nrow(grid))

for (k in 1:nrow(grid)) {
 n <- grid[k, 1]
 d <- grid[k, 2]
 X <- replicate(d, rnorm(n))
 cat("n = ", n, ", d = ", d, ", time = ", sep = "")
 times[k] <- median(microbenchmark(sigma_2_besser(X, rep(1, d)), times = 1000)$time)
 cat(times[k] / 1e9, "\n")
}
```

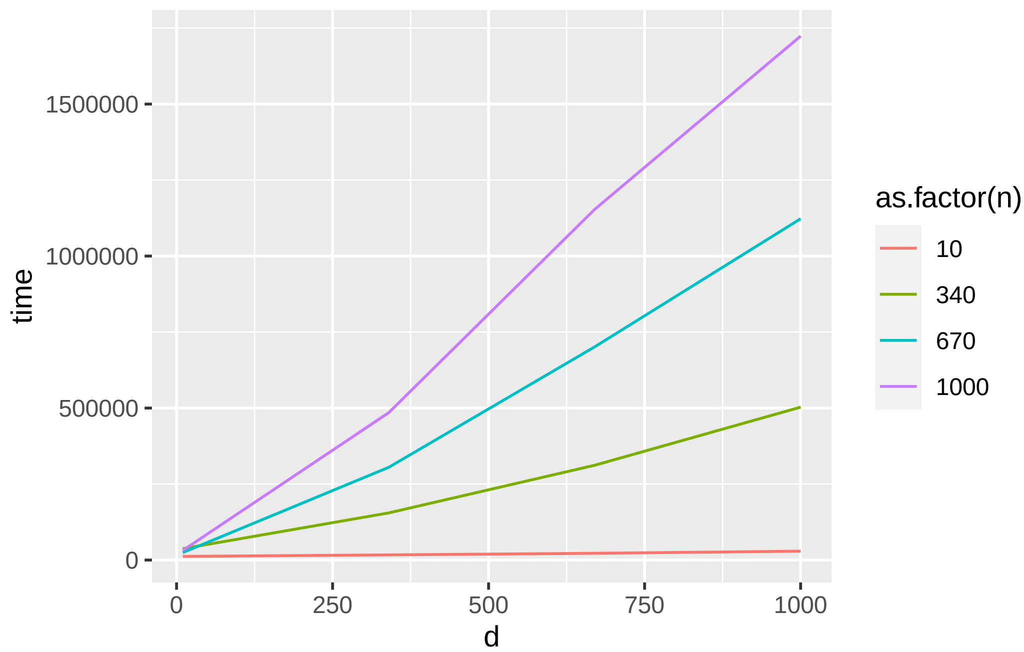
```
n = 10, d = 10, time = 1.19955e-05
n = 340, d = 10, time = 3.728e-05
n = 670, d = 10, time = 2.4525e-05
n = 1000, d = 10, time = 3.1592e-05
```

```
n = 10, d = 340, time = 1.69285e-05
n = 340, d = 340, time = 0.000155073
n = 670, d = 340, time = 0.000304966
n = 1000, d = 340, time = 0.0004847685
n = 10, d = 670, time = 2.2245e-05
n = 340, d = 670, time = 0.00031182
n = 670, d = 670, time = 0.000701359
n = 1000, d = 670, time = 0.001153717
n = 10, d = 1000, time = 2.91165e-05
n = 340, d = 1000, time = 0.000502958
n = 670, d = 1000, time = 0.001122692
n = 1000, d = 1000, time = 0.001723579
```

```r
cbind(grid, time = times) |>
 ggplot(aes(n, time, color = as.factor(d))) +
 geom_line()
```




```r
cbind(grid, time = times) |>
 ggplot(aes(d, time, color = as.factor(n))) +
 geom_line()
```




Wir sehen also, dass die Laufzeit tatsächlich linear in $d$ und $n$ ist.