Programmieren mit Statistischer Software **11. Block**
Grün **SS 2017**

11 Effizientes Programmieren

Aufgabe 1:

Schreiben Sie eine Funktion,

```
> rpoisson <- function(n, lambda) { }
```

die `n` poissonverteilte Zufallsvariablen mit Parameter λ zieht und daraus den Mittelwert bestimmt und zurückgibt. Überprüfen Sie die Argumente und achten Sie auf Lesbarkeit und Effizienz.

Rufen Sie die Funktion jeweils 5000 Mal auf und variieren Sie `n` mit den Werten 10, 100, 1000, 10000 und setzen Sie `lambda` auf 0.1, 1, 10, 100. Welchen empirischen Mittelwert und empirische Varianz haben diese jeweils 5000 Beobachtungen? Was sind die theoretischen Werte?

Aufgabe 2:

Schreiben Sie eine Funktion `sillyzv()`, die die folgenden Argumente hat:

- `n`: Skalar, der die Anzahl der Beobachtungen pro Variable angibt.
- `k`: Skalar, der die Anzahl der Variablen angibt.
- `mean`: Vektor der Länge `k` (oder geeignet wiederholbar), der die Mittelwerte der Variablen enthält.
- `sd`: Vektor der Länge `k` (oder geeignet wiederholbar), der die Standardabweichung der Variablen enthält.

Die Funktion soll entsprechend Beobachtungen aus der Normalverteilung ziehen und eine Matrix zurückgeben, die `n` Zeilen und `k` Spalten hat. D.h. die Mittelwerte und Standardabweichung in den Spalten variieren je nach `mean` und `sd`. Überprüfen Sie die Argumente und achten Sie auf Lesbarkeit.

- Versuchen Sie `sillyzv()` möglichst ineffizient zu programmieren. Sie sollen dabei jedoch keine unnötige Dinge tun und die n Beobachtungen pro Variable mit einem Aufruf erzeugen.

Hinweis: In der Datei `Transcript-11.pdf` gibt es Beispiele für ineffizientes Programmieren.

Aufgabe 3:

Schreiben Sie eine Funktionen `smartzv()`, die dieselben Argumente und dasselbe Rückgabeobjekt wie die Funktion `sillyzv()` aus der vorhergehenden Aufgabe hat.Überprüfen Sie die Argumente und achten Sie auf Lesbarkeit.

- Versuchen Sie `smartzv()` möglichst effizient zu programmieren. Achten Sie darauf, dass die Ergebnisse gleich denen von `sillyzv()` sind.
- Vergleichen Sie die Zeiten für den Aufruf und überprüfen Sie die Gleichheit der Resultate:

```
> set.seed(0606)
> t1 <- system.time({x1 <- sillyzv(10^4, 100, 1:100, 1:100)})
> set.seed(0606)
> t2 <- system.time({x2 <- smartzv(10^4, 100, 1:100, 1:100)})
> t1 / t2
> all.equal(x1, x2)
```

Aufgabe 4:

Beim **Ziehen mit Zurücklegen**, d.h., wenn aus einem Datensatz der Größe n n-mal mit Zurücklegen gezogen wird, werden manche Elemente mehrfach und manche Elemente gar nicht gezogen. Schreiben Sie eine Funktion

```
> srcount <- function(n, b) { ... }
```

welche `b` Ziehungen für einen Datensatz der Größe `n` (mit `n` unterschiedlichen Elementen) durchführt und jeweils die Anzahl der unterschiedlichen Elemente pro Ziehung berechnet. Rückgabewert der Funktion ist der Vektor der `b` Häufigkeiten von unterschiedlichen Elementen. Überprüfen Sie die Argumente und achten Sie auf Lesbarkeit und Effizienz.

Berechnen Sie die Häufigkeiten für einen Datensatz der Größe `n = 6325` bei `b = 4000` Durchläufen.

Aufgabe 5:

Laden Sie den Datensatz `genotype` aus dem Paket **MASS**

- Wie viele Beobachtungen und Variablen hat der Datensatz?
- Bestimmen Sie die Anzahl an Beobachtungen für jede Kombination an Genotypen der leiblichen und aufziehenden Mutter.
- Wie groß ist das durschnittliche Gewicht für jede Kombination an Genotypen der leiblichen und aufziehenden Mutter?
- Visualisieren Sie den Datensatz geeignet, um zu analysieren, ob der Genotype der leiblichen oder der aufziehenden Mutter einen stärkeren Einfluss auf das durchschnittliche Gewicht haben.