Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(7) Kontrollstruktur und Schleifen

Literaturhinweise

Wir folgen Cotton (2013), Kapitel 8 und Kapitel 9, sowie Wickham (2019), Kapitel 5.

Überblick

Perse wird Programmiercode streng sequentiell Befehl für Befehl ausgeführt.

Manchmal möchte man von dieser rein sequentiellen Befehlsreihenfolge abweichen.

Die prinzipiellen Werkzeuge dafür sind **Kontrollstrukturen** und **Schleifen**

- Kontrollstrukturen bedingen die Ausführung von Befehlen
    - R bietet **if()** und **switch()** als Kontrollstrukturen
- Schleifen erlauben die adaptive Wiederholung von Codeabschnitten
    - R bietet **for()**, **while()**, und **repeat()** Loops

Kontrollstruktur und Schleifen

- if-statements
- for-loops
- Übungen und Selbstkontrollaufgaben

Kontrollstruktur und Schleifen

- **if-statements**
- for-loops
- Übungen und Selbstkontrollaufgaben

if-statements

```
if (Bedingung){
  TrueAktion
}
```

Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` evaluiert.

Wenn `Bedingung` `FALSE` ist, wird `TrueAktion` nicht evaluiert.

if-else-statements

```
if (Bedingung){
  TrueAktion
} else {
  FalseAktion
}
```

Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` evaluiert.

Wenn `Bedingung` `FALSE` ist, wird `FalseAktion` evaluiert.

Beispiele

```
x = 1
if(x > 0){
  print("x ist größer als 0")
}
```

```
> [1] "x ist größer als 0"
```

```
y = 1
if(y > 0){
  print("y ist größer als 0")
} else{
  print("y ist nicht größer als 0")
}
```

```
> [1] "y ist größer als 0"
```

Logische Operatoren in R

Die Boolesche Algebra und R kennen zwei logische Werte: TRUE und FALSE

Bei Auswertung von Relationsoperatoren ergeben sich logische Werte

Relationsoperator	Bedeutung
==	Gleich
!=	Ungleich
<, >	Kleiner, Größer
<=, >=	Kleiner gleich, Größer gleich
|	ODER
&	UND

<, <=, >, >= werden zumeist auf numerische Werte angewendet.

==, != werden zumeist auf beliebige Datenstrukturen angewendet.

| und & werden zumeist auf logische Werte angewendet.

Die Funktion xor() implementiert das exklusive ODER.

Beispiele

```
x = 1
y = 2
```

```
# Test auf Gleichheit
if(x == y){
  print("x ist gleich y")
} else {
  print("x ist ungleich y")
}
```

```
> [1] "x ist ungleich y"
```

```
# Test auf Ungleichheit
if(x < y){
  print("x ist kleiner als y")
} else {
  print("x ist gfößer oder gleich y")
}
```

```
> [1] "x ist kleiner als y"
```

Beispiele

```
x = 2
y = 2
```

```
# logisches UND/ODER
if(x == y | x < y){
  print("x ist kleiner oder gleich y")
} else {
  print("x ist größer als y")
}
```

```
> [1] "x ist kleiner oder gleich y"
```

```
# logisches UND
if(x > y & x != y){
  print("x ist größer als y")
} else {
  print("x ist kleiner oder gleich y")
}
```

```
> [1] "x ist kleiner oder gleich y"
```

Fehlerhafte Bedingungen

Die Bedingung sollte einem einzigen logischen Wert entsprechen

```
if("x") 1
```

```
> Error in if ("x") 1: Argument kann nicht als logischer Wert
> interpretiert werden
```

```
if(NA) 1
```

```
> Error in if (NA) 1: Fehlender Wert, wo TRUE/FALSE nötig ist
```

```
if(c(T,F)) 1
```

```
> Warning in if (c(T, F)) 1: Bedingung hat Länge > 1 und nur das erste
> Element wird benutzt
```

```
> [1] 1
```

Motivation

Kombinierte if-else Statements werden leicht unübersichtlich

```
x = 2
if (x == 1){
  print("Aktion 1")
} else if(x == 2){
  print("Aktion 2")
} else if(x == 3){
  print("Aktion 3")
} else if(x == 4){
  print("Aktion 4")
}
```

```
> [1] "Aktion 2"
```

switch-statement mit Integervariable

Bei switch Variable *i* vom Typ Integer wird die *i*te Aktion ausgeführt.

```
x = 2                                                    # switch Variable
switch(x,                                                # x = 2
print("Aktion 1"),                                       # 1. Aktion
print("Aktion 2"),                                       # 2. Aktion
print("Aktion 3"),                                       # 3. Aktion
print("Aktion 4"))                                       # 4. Aktion
```

```
> [1] "Aktion 2"
```

switch-statement mit Charaktervariable

Bei switch Variable **x** vom Typ Character wird die Aktion mit Namen **x** ausgeführt.

```
x = "a"                                          # switch Variable
switch(x,                                        # x = 2
a = print("Aktion a"),                           # a Aktion
b = print("Aktion b"),                           # b Aktion
c = print("Aktion c"),                           # c Aktion
d = print("Aktion d"))                           # d Aktion
```

```
> [1] "Aktion a"
```

if vs. switch-statements

- Jedes if-statement kann als switch-statement formuliert werden
- Jedes switch-statement kann als if-statement formuliert werden
- if-statements sind in der Anwendung häufiger
- if-else-statements sollten nicht zu komplex designed werden
- Bei komplexe if-else-statements kann sich eine Formulierung als switch lohnen

Kontrollstruktur und Schleifen

- if-statements
- **for-loops**
- Übungen und Selbstkontrollaufgaben

for-loops

Generelle Form

```
for (item in vector) perform_action
```

- `perform_action` wird für jedes `item` in `vector` einmal evaluiert
- Der Wert von `item` wird jeweils aktualisiert.

Beispiel

```
for (i in 1:3){
  print(i)
}
```

```
> [1] 1
> [1] 2
> [1] 3
```

Simulationsbeispiele

Typischerweise wird innerhalb eines for-loops etwas erzeugt und gespeichert

- Die entsprechende Speicherstruktur sollte vorallokiert werden
- Der entsprechende Arbeitsspeicher ist schon bereitgestellt (Speed)
- Fehler können leichter detektiert werden (NaNs)

```
ns    = 3                          # Anzahl an Simulationen
X_bar = rep(NaN, ns)               # Speicherstruktur

# Simulationsiterationen
for(i in 1:ns){                    # Iterationsindices sind typischerweise i,j,k
  X         = rnorm(12)            # Realisierung von 12 Z-Variablen
  X_bar[i]  = mean(X)              # Mittelwert der i-ten Realisierung
  print(X_bar)                     # Anzeige zur Demonstration
}
```

```
> [1] -0.362    NaN    NaN
> [1] -0.362 -0.133    NaN
> [1] -0.362 -0.133  0.121
```

Simulationsbeispiele

Typischerweise ändern sich innerhalb eines for-loops Parameter

- `seq_along` sollte zur linearen Indizierung genutzt werden

```
mu    = c(0,5,50)                    # Drei Erwartungswertparameter
X_bar = rep(NaN, ns)                 # Speicherstruktur

# Simulationsiterationen
for(i in seq_along(mu)){             # Iterationsindices sind typischerweise i,j,k
  X         = rnorm(12, mu[i], 1)    # Realisierung von 12 N(\mu,1) Varialben
  X_bar[i]  = mean(X)                # Mittelwert der i-ten Realisierung
  print(X_bar)                       # Anzeige zur Demonstration
}
```

```
> [1] 0.127   NaN   NaN
> [1] 0.127 4.957   NaN
> [1]  0.127  4.957 49.827
```

Simulationsbeispiele

for-loops können auch geschachtelt werden

Für **jede** Iteration der äußeren Schleife werden **alle** Iterationen der inneren Schleife evaluiert

```
n_i = 2                                 # Anzahl der Iterationen äußere Schleife
n_j = 3                                 # Anzahl der Iterationen innere Schleife
for(i in 1:n_i){                        # Äußere Schleife
  for (j in 1:n_j){                     # Innere Schleife
    print(c(i,j))                       # Anzeigen der Iternationsindices [i,j]
    }
}
```

```
> [1] 1 1
> [1] 1 2
> [1] 1 3
> [1] 2 1
> [1] 2 2
> [1] 2 3
```

Beispiel

Definition (Schätzerkonsistenz)

$X_1, ..., X_n \sim p_\theta$ sei die Stichprobe eines parametrischen statistischen Produktmodells $\mathcal{M}$ und $\hat{\tau}_n$ sei ein Schätzer von τ. Eine Folge von Schätzern $\hat{\tau}_1, \hat{\tau}_2, \ldots$ wird dann eine *konsistente Folge von Schätzern* genannt, wenn für jedes $\epsilon > 0$ und jedes $\theta \in \Theta$ gilt, dass

$$\lim_{n \to \infty} \mathbb{P}_\theta \left(|\hat{\tau}_n(X) - \tau(\theta)| \geq \epsilon \right) = 0.$$

Wenn $\hat{\tau}_1, \hat{\tau}_2, \ldots$ eine konsistente Folge von Schätzern ist, dann heißt $\hat{\tau}_n$ *konsistenter Schätzer*.

$\mathbb{P}_\theta$ hängt von ϵ und n ab.

Beispiel

```
# Definitionen
mu      = 1                                          # w.a.u. Erwartungswertparameter
sigsqr  = 2                                          # w.a.u. Varianzparameter
n       = seq(1,1e3,by = 10)                         # Stichprobengröße n
eps     = c(0.15, 0.10, 0.05)                        # \epsilon Werte
ne      = length(eps)                                # Anzahl \epsilon Werte
nn      = length(n)                                  # Anzahl Stichprobengrößen
ns      = 1e1                                        # Anzahl der Simulationen
E       = array(rep(NaN,nn,ne,ns), dim = c(nn,ne,ns))  # Ereignisindikator Array

# Simulationen
for(e in seq_along(eps)){                            # \epsilon Wert Iterationen
  for(i in seq_along(n)){                            # Stichprobengrößeniterationen
    for(s in 1:ns){                                  # Simulationsiterationen
      x   = rnorm(n[i], mu, sqrt(sigsqr))            # Stichprobenrealisationen
      if(abs(mean(x) - mu) >= eps[e]){               # |X_bar - \mu)| \ge \epsilon
                E[i,e,s] = 1                         # Ereignisindikator
      } else {                                       # |X_bar - \mu)| < \epsilon
                E[i,e,s] = 0                         # Ereignisindikator
            }
        }
    }
}
```

E kann zum Schätzen von $\mathbb{P}_\theta$ gemittelt werden.

for-loop Alternativen

while-loops iterieren Codeabschnitte basierend auf einer Bedingung

```
while(condition){
  TrueAktion          # TrueAktion wird ausgeführt, solange condition == TRUE
}
```

repeat-loops wiederholen Codeabschnitte bis zu einem 'break' Befehl

```
repeat{
  Aktion              # Aktion wird ausgeführt, bis ein break Befehl evaluiert wird
}
```

for-loops können als while-loops reformuliert werden, while-loops als repeat-loops

Generell wird in R die weniger flexible **apply()** Funktionalität bevorzugt.

Kontrollstruktur und Schleifen

- if-statements
- for-loops
- **Übungen und Selbstkontrollaufgaben**

Übungen und Selbstkontrollaufgaben

1. Erzeugen Sie mithilfe eines for-loops eine 4 x 5 Matrix, deren Elemente gleich den Spaltenindizes sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    2    3    4    5
> [2,]    1    2    3    4    5
> [3,]    1    2    3    4    5
> [4,]    1    2    3    4    5
```

2. Erzeugen Sie mithilfe eines for-loops eine 4 x 5 Matrix, deren Elemente gleich den Zeilenindizes sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    1    1    1    1
> [2,]    2    2    2    2    2
> [3,]    3    3    3    3    3
> [4,]    4    4    4    4    4
```

3. Erzeugen Sie mithilfe zweier geschachtelter for-loops und eines if-statements eine 4 x 5 Matrix, deren Elemente gleich den Spaltenindizes sind, wenn der jeweilige Spaltenindex größer oder gleich dem Zeilenindex ist, und deren Elemente ansonsten Null sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    2    3    4    5
> [2,]    0    2    3    4    5
> [3,]    0    0    3    4    5
> [4,]    0    0    0    4    5
```

References

Cotton, Richard. 2013. *Learning R*. First Edition. Beijing ; Sebastopol, CA: O'Reilly.

Wickham, Hadley. 2019. *Advanced R, Second Edition*. CRC Press.