Methoden der Ökonometrie

Einführung findet im Cip-Pool statt! (RZ 1)

Dieses Blatt umfasst 17 Seiten und dient als Kurzeinführung zum bereits vom Lehrstuhl parallel angebotenen Kurs Programmieren mit R. Im Anschluss an die Einführung werden noch Trainingsaufgaben besprochen; der Code zu den Eigenaufgaben wird nach der Einführung online gestellt.

Installation von `R` (Beispiel für Windows)
`R` ist eine freie Statistik- und Ökonometrie-Software, genauer gesagt eine Interpretersprache zu diesem Zweck. Installationsdatei, Ergänzungspakete und sonstige Informationen, wie z.B. Manuals, können unter http://www.r-project.org/ heruntergeladen werden. Um zur .exe-Datei zu gelangen klicken Sie bei Download auf `CRAN`, z.B. bei Germany auf `http://cran.rakanu.com/`, bei Download and Install `R Windows`, anschließend auf `base` und laden Sie die entsprechende .exe-Datei herunter und führen Sie diese aus. Da Sie im weiteren Verlauf mit `R` auf Ihrem Rechner arbeiten sollten, empfiehlt es sich `R` zusammen mit einer „Oberfläche/GUI“ (z. B. R Studio) oder einem Editor (z. B. Notepad ++ mit dem Plug-In NppToR) zu installieren.

Einarbeitung in `R`
Einführende Dokumentationen finden Sie beispielsweise auf der `R`-Homepage unter R-intro.pdf oder R-lang.pdf. Sonstige auf der `R`-Homepage verlinkte Dokumentationen sind z.B. eine deutsche Einführung, eine englische Einführung oder ein kurzes Befehlsverzeichnis. Außerdem gibt es ein wikibook zu `R`.

Hinweise zur Programmierung in `R`

Die im Folgenden vorgestellten Befehle finden Sie auch in diesem `R`-Skript, das im Anschluss an die folgenden Erläuterungen angefügt ist.

1. **Allgemeine Hinweise**

   - Wenn Sie Ergebnisse speichern bzw. bereits bestehende Dateien verwenden möchten, wechseln Sie zunächst in das entsprechende Verzeichnis (in `R`: `Datei`, `Verzeichnis wechseln` oder alternativ mit dem Befehl `getwd()` können Sie das aktuelle Arbeitsverzeichnis anzeigen lassen bzw. mit `setwd()` setzen; achten Sie dabei, dass sie statt \ einen doppelten Backslash \\ oder einen einfachen Slash / zur Trennung der Ordnerstrukturen benutzen).
   - Befehle können direkt in die Konsole eingegeben werden oder über ein Skript (`Datei`, `Neues Skript`, abspeichern mit der Endung `.R`). Befehle werden in der Konsole über

`Return` bestätigt, in einem Skript durch Markieren des auszuführenden Teils und drücken von `Strg + R`. Insbesondere für längere Programmcodes, die später noch benötigt werden, empfiehlt es sich **immer**, ein `Skript` zu verwenden.
Für Skripte können natürlich oben genannte Editoren benutzt werden, die per entsprechendem Tastenkürzel (`R`-Studio: `Strg + R` / `Strg + Enter`, Notepad++: manuell) das Skript ausführen.

- Groß- und Kleinschreibung sind zu unterscheiden.
- Das Dezimaltrennzeichen ist ein Punkt. (Bitte bei der Einlese von Daten darauf achten!)
- Kommentare können in `R` mit # eingefügt werden.
- Hilfe zu konkreten Befehlen erhält man mit `?`, z.B. `?lm`, Hilfe zu Stichworten mit `??` oder `help.search(≪Suchbegriff(e)>")`, z.B. `??regression` oder `help.search("linear model")`.
- `R` kann auch als Taschenrechner verwendet werden, z.B. durch Eingabe von `1+2` in die Konsole.
- Variablen können mit Hilfe der Syntax `<-` Werte zugeordnet werden, die anschließend wieder abgerufen werden können (sofortige Ausgabe durch in Klammern setzen), z.B.
  ```
  a <- 1
  b <- 2
  (c <- a + 2*b)
  ```
  ergibt sich für `c` als Ausgabe der Wert `5`.
  Anmerkung: Inzwischen ist es auch möglich, Variablen mit = zu definieren.
- Bei zu langen Befehlen, die einen Zeilenumbruch erfordern, muss dieser so gesetzt werden, dass der Teil bis zum Umbruch keinen vollständigen Befehl ergibt und somit noch nicht ausgeführt werden kann. Betrachten Sie zu dieser Problematik beispielsweise
  ```
  s1 <- 1 + 2 +
  3
  ```
  im Vergleich zu
  ```
  s2 <- 1 + 2
  + 3
  ```
  bzw. zu
  ```
  s3 <- (1 + 2
  + 3)
  ```
- Bereits ausgeführte Befehle können mit der ↑-Taste in der Konsole wieder geholt werden.
- Mit `ls()` wird angezeigt, welche Objekte (z.B. Vektoren, Matrizen) sich im Workspace befinden. Diese können einzeln mit `save()` bzw. gesamt mit `save.image()` gespeichert werden.
- Mit dem Befehl `rm(<name>)` können Sie sich im Workspace befindende Objekte löschen.
- Gewöhnen Sie sich von Anfang an eine übersichtliche Programmiertechnik an und kommentieren Sie Ihre Programme sorgfältig (vgl. Google's R-Style Guide).

2. **Atomare Datentypen**

   `R` besitzt sechs atomare Datentypen, `logical` (`TRUE`, `FALSE`, `NA`), `integer` (ganze Zahlen), `double` (beliebig numerische Zahlen, die mehr Speicherplatz als integers benötigen), `complex` (Komplexe Zahlen), `character` (Zeichenfolgen) und `raw`.

```
typeof(3.4)
typeof("3.4")
class(3.4)
class("3.4")
```

3. **Vektoren und Matrizen**

   Vektoren und Matrizen sind Sammlungen oben genannter, atomarer Datentypen mit einem Attribut: bei Vektoren das Attribut Länge `length()`, bei Matrizen die Dimension `dim()`.

   - Vektoren werden in R üblicherweise über den Befehl

     `<name> <- c(...)`

     eingegeben. Der Vektor kann über die Eingabe von `<name>` betrachtet werden. Vektoren werden immer dimensionslos abgespeichert (d. h. sind weder Spalten- noch Zeilenvektor).
     Beispiel:

```
vec1 <- c(1,3,5,7)
vec1
```

     bzw.

```
(vec1 <- c(1,3,5,7))
```

     Alternative Möglichkeiten zur Eingabe von Vektoren sind beispielsweise

```
vec2 <- 1:4
vec3 <- rep(1, times=7)
vec4 <- rep(c(1,2), times=2)
vec5 <- rep(c(1,2), each=2)
vec6 <- c(c(1,2,3), vec1)
```

   - Matrizen werden über den Befehl

     `<name> <- matrix(c(...), nrow=<Zeilenanzahl>, ncol=<Spaltenanzahl>)`

     eingegeben, wobei zu beachten ist, dass genau so viele Einträge übergeben werden müssen wie Zeilen*Spalten. Die Eingabe erfolgt spaltenweise, außer man wählt die Option `byrow=TRUE`.
     Beispiele:

```
mat1 <- matrix(1:4)
mat2 <- matrix(c(1,2,3,4), ncol=2)
mat3 <- matrix(c(1,2,3,4), ncol=2, byrow=TRUE)
```

     Alternativ können Matrizen auch aus Vektoren erstellt werden:

```
mat4 <- matrix(c(vec1,vec2), ncol=2)
```

`mat5 <- matrix(c(1:3,c(1,2,3)), ncol=2)`

Es reicht also bei einer festen Anzahl an Einträgen, die Spalten- oder Zeilenzahl anzugeben. Für Matrizen mit lauter gleichen Einträgen, müssen die Spalten- und die Zeilenzahl angegeben werden, z.B.

`mat6 <- matrix(NA, ncol=3, nrow=2)`

- Numerische Indizierung: Um einzelne Elemente von Matrizen oder Vektoren aufrufen zu können, werden der Zeilen- und der Spaltenindex in eckigen Klammern angegeben. Um ganze Zeilen aufzurufen, wird der Spaltenindex weggelassen (analog für ganze Spalten). Beispiele:

  `mat3[2,1]`

  `mat3[2,]`

  `mat3[,1]`

  `vec6[3]` Ebenso können einzelne Elemente einer Matrix verändert werden:

  `(mat6[1,2] <- 3)`

- Wichtig: Reduziert man mehrdimensionale Objekte auf niedrig dimensionalere Strukturen (z. B. Array $\Longrightarrow$ Matrix $\Longrightarrow$ Vektor), so geht die Information der zusätzlichen Dimension im Fall „Matrix $\Longrightarrow$ Vektor“ veloren. Ausweg: `drop = FALSE`:

  `mat1[ ,1]`

  `mat1[ ,1, drop = FALSE]`

- Rechenoperationen wie `+`, `*`, `^` und auch logische (s.u.) werden bei Vektoren und Matrizen elementweise ausgeführt.

- Matrixmultiplikation funktioniert mit `%*%`, transponieren erfolgt durch `t()`. Man beachte aber, dass durch die Dimensionslosigkeit der Vektoren `R` diese manchmal transformiert (vgl. Fall 2 und 3) werden:

  `vec1 * vec2 # Elementweise Multiplikation: [3x1] * [3x1] = [3x1]`

  `vec1 %*% vec2 # Skalarprodukt <vec1, vec2>: [1x3] * [3x1] = [1x1]`

  `t(vec1) %*% vec2 # Skalarprodukt` $\texttt{vec1}^T\texttt{vec2}$`: [1x3] * [3x1] = [1x1]`

  `vec1 %*% t(vec2) # Äußeres Produkt vec1` $\texttt{vec2}^T$`: [3x1] * [1x3] = [3x3]`

- Die Länge eines Vektors $\mathbf{x}$ kann mit `length(x)` abgefragt werden, die Dimension der Matrix $\mathbf{M}$ mit `dim(M)` bzw. `nrow(M)` und `ncol(M)`.

- Es können beispielsweise auch Vektoren mit unterschiedlicher Länge addiert werden. Dann wiederholt `R` den kürzeren Vektor so lange, bis er die Länge des anderen hat.

  Beispiel:

  `vec7 <- rep(1, times = 8) + 1:3`

  `vec8 <- c(1,2,3,4) + 3`

  Bei `vec7` gibt R zwar eine Warnung aus, führt die Rechnung aber trotzdem durch! Also Vorsicht bei längeren Programmen!

4. **Logische Operatoren, `if`-Befehl, `for`-Schleife, Funktionen**

- Die logischen Operatoren sind `<` (kleiner), `<=` (kleiner gleich), `>` (größer), `>=` (größer gleich), `==` (gleich) und `!=` (ungleich).
  Als Ausgabe erhält man `TRUE` oder `FALSE`, bzw. in Berechnungen wird `TRUE` der Wert 1 und `FALSE` der Wert 0 zugeordnet.

- Sollen bestimmte Befehle nur in manchen Fällen durchgeführt werden, kann mit Hilfe der logischen Operatoren eine `if`-Abfrage durchgeführt werden.
  Der Befehl ist

```
if (<Bedingung>)  {
    <Befehl>
}
```

  Der `if`-Befehl kann noch um einen `else`-Teil erweitert werden

```
else {
     <Befehl>
}
```

  Die Befehle in der `if`-Abfrage müssen entweder in einer Zeile oder in geschweiften Klammern geschrieben werden.
  Beispiele:

```
if (1==1) {"richtig"} else {"falsch"}
if (1==2) {"richtig"} else {"falsch"}
d <- 5
if (d>=0) {sqrt(d)} else {"positive Zahl übergeben"}
```

- Soll ein Befehl mehrmals wiederholt werden, kann eine `for`-Schleife erstellt werden. Der Befehl dazu ist

```
for (<Bedingung>)  {
    <Befehl>
}
```

  Beispiel:

```
n1 <- 0
for (i in 1:6) { n1 <- n1 + i }
n1
```

- Der `if`-Befehl und die `for`-Schleife können auch kombiniert werden.
  Beispiel:

```
for (i in 1:6){
    if (i == 1){
      n2 <- i
```

```
    } else{
      n2 <- n2 + i
    }
}
n2
```

- Sie können sich auch eigene Funktionen definieren. Dies geschieht mit

```
<name> <- function(<Argument(e)>) {
     <Befehl>
 }
```

  Beispiel:

```
XdurchY <- function(x,y=1){
     if (y==0)
       print("Zweiter Eintrag darf nicht 0 sein!")
     else
       x / y
}
XdurchY(28,4)
XdurchY(28)
XdurchY(28,0)
```

  Dabei wird hier 1 als Default-Wert für `y` in der Funktion `XdurchY` gewählt.

5. **Zufallszahlen**

- In R können auch Zufallszahlen aus vielen Verteilungen erzeugt werden.
  Eine Übersicht zu Zufallszahlen und Verteilungen finden Sie in der R-intro als Kapitel 8.1.
  Um die Ergebnisse wieder reproduzieren, bzw. mit denen anderer vergleichen zu können, ist es nötig, einen Startwert für die Erzeugung von Zufallszahlen festzusetzen. Dies geschieht mit dem Befehl `set.seed(<ganze Zahl>)`.
  Beispiel:

```
set.seed(42)
x <- runif(50, min=2, max=7)
y <- 14 + 3*x + rnorm(50, mean=0, sd=4)
```

6. Daten einlesen

- Bereits erstellte Datensätze können in R eingelesen werden. Dies geschieht beispielsweise mit dem Befehl

  ```
  <name> <- read.table("<Datei>.txt", header = TRUE, sep="")
  ```

  Dabei werden die Daten aus der Textdatei `<Datei>` im R-Workspace zusammen unter `<name>` abgespeichert. Auf die einzelnen Variablen kann nach Ausführung des Befehls `attach(<name>)` über deren Namen zugegriffen werden, da wegen `header = TRUE` auch die Bezeichnungen der Variablen übernommen werden (diese müssen dazu bereits in der .txt-Datei enthalten sein).
  Die Option `sep=""` ist die default-Einstellung und funktioniert für Leerzeichen, Tabstopps, usw. als Trennzeichen. Ist das Trennzeichen z.B. ein Komma, muss `sep=","` gesetzt werden.
  Bevor die Datei eingelesen werden kann, muss in R über `Datei`, `Verzeichnis wechseln` zunächst der Ordner ausgewählt werden, in dem sich die Datei befindet, oder es wird der komplette Pfad in Anführungszeichen eingegeben.
  Weitere Möglichkeiten zum Einlesen von Daten finden Sie beispielsweise in Kapitel 7 der R-intro.

7. OLS-Regression

- Lineare Regressionsmodelle werden über den `lm`-Befehl geschätzt. Das lineare Einfachmodell von y auf x mit Konstante ist beispielsweise gegeben durch

  ```
  lm(y ~ 1 + x)
  ```

  Dabei müsste die Konstante nicht explizit angegeben werden, allerdings müsste `- 1` geschrieben werden, wenn keine Konstante enthalten sein soll. Mit `summary` wird eine Zusammenfassung der Regressionsergebnisse ausgegeben, z.B. `summary(lm(y ~ x))`.
  Weitere Hinweise (z.B. welche Informationen aus der Schätzung gewonnen werden können) finden Sie in der R-intro in den Kapiteln 11.1 bis 11.3.
  Beispiel:

  ```
  ols <- lm(y ~ 1 + x)
  summary(ols)
  plot(x,y)
  abline(ols)
  u_hat <- residuals(ols)
  y_hat <- predict(ols)
  hist(u_hat)
  ```

8. Graphiken

- Daten können in R auch graphisch dargestellt werden. Dazu gibt es sehr viele Möglichkeiten. Ein wichtiger Befehl ist der `plot`-Befehl, der ein x-y-Diagramm erstellt.

  `plot(<x>,<y>,<Optionen>)`

  Äußerst einfaches Beispiel:
  `plot(vec1,vec2)`
  Mögliche Optionen finden Sie in der R-intro als Kapitel 12.1.4.
  In ein bestehendes x-y-Diagramm können zusätzlich Punkte (`points`), Linien (`lines`, `abline`, ...), Texte (`text`), ... eingetragen werden.
  Eine Liste dieser und weiterer Möglichkeiten finden Sie ebenfalls in der R-intro als Kapitel 12.2 und Optionen dazu in 12.5.1.
  Beispiel:
  `plot(x,y)`

- Zum Speichern der Graphiken können beispielsweise die Befehle `pdf(file="name.pdf")` oder `postscript(file="name.eps")` verwendet werden. Bei beiden wird ab dem entsprechenden Befehl so lange in die pdf- bzw. eps-Datei geplottet bis der Befehl `dev.off()` gegeben wird.

9. Pakete installieren und laden

- Viele Funktionen sind noch nicht in der Basis-Ausstattung von R enthalten. Diese müssen installiert und geladen werden bevor sie genutzt werden können.
- Einige Pakete sind schon installiert aber nicht geladen. Über

  `library(≪package name>")`

  können sie zugänglich gemacht werden.
- Andere Pakete müssen erst installiert werden. Bei Internet-Zugriff kann dies mit Hilfe von `Pakete`, `Installiere Paket(e)...` oder alternativ in der Console über den Befehl `install.packages(≪package name>")` geschehen. Ohne Internet-Zugriff müssen die Pakete zuerst auf den Rechner geholt werden und anschließend über `Pakete`, `Installiere Paket(e) aus lokalen Zip-Dateien...` installiert werden. Um die Pakete nutzen zu können ist wieder der Befehl `library` nötig. Einmal installierte Pakete bleiben auf dem Rechner installiert, müssen aber bei jeder neuen Anwendung mit `library(<package name>)` wieder geladen werden.

10. **Fazit: Was sollten Sie nun verstanden haben bzw. können?**

- `R` ist eine Interpretersprache mit der Konsole als Kommunikationsmittel, an die Befehle oder Skriptdateien zur Ausführung geschickt werden und Fehler zurückgegeben werden können.
- Setzen des Working Directories (Arbeitspfad), Speichern (einzelner Variablen) des Working Spaces (Arbeitsumgebung; Datei im .RData-Format), Ausführen externer `R`-Skripte (Datei im .R-Format), Einlesen externer Datensätze und Löschen einzelner bzw. aller Variablen.
- Matrizenoperationen, d. h. Addition, (komponentenweise und Matrizen-) Multiplikation, Skalarprodukt, Transponierung, Invertierung, Spur und Determinante.
- Ansprechen der Komponenten von Vektoren, Matrizen und Datensätzen mittels (positiv und negativ) numerischer, logischer und namentlicher Indizierung.
- Benutzen des `lm()`-Befehls für Regressionen mit stetigen und kategoriellen Variablen, mit und ohne Konstante, mit Interaktionen und Transformationen der Variablen.

```
# Methoden der Ökonometrie
# R-Einführung
# Skript, das die Befehle der Angabe enthält

#####################################################################
# 1. Allgemeine Hinweise
#####################################################################

# Hilfe
# Hilfe zu R-Befehlen
?lm
# Hilfe zu Stichwörtern
??regression
# bzw. das gleiche mit dem Befehl help.search
help.search("linear model")

#was ist das Arbeitsverzeichnis
getwd()
# wir können das Arbeitsverzeichnis neu setzen, z. B. setwd("G:\\R")
# Pfade sind Namen, gehören also in Anführungszeichen;
# R kennt - wie viele andere Systeme auch - \ nicht als Trennungszeichen,
# deswegen muss man entweder zwei Backslashes oder einen Slash / machen
setwd("G:/R") # für das G-Laufwerk

# Setzen von Variablen:
a <- 1
a = 0
# diese sind jetzt im Workspace gespeichert; man sieht dies mit
ls()
# will man diese Variable löschen, so benutzt man den Befehl
rm(a)
ls()
# um Speicher zu sparen, kann man vor neuen Aufgaben den gesamten Workspace löschen
rm(list=ls(all=TRUE))

# will man ihn speichern, so kann man ihn in einen namen mit Kürzel .RData speichern
save.image()   # Workspace speichern, z.B.: save.image("Workspace.RData").
load() # Laden von Dateien (auch des Workspace).

# Nun zu gängigen Rechnungen
```

```
a <- 2
b <- 2
(c <- a + 2*b)

s1 <- 1 + 2 +
  3
s2 <- 1 + 2
  + 3
s3 <- (1 + 2
  + 3)
s1
s2
s3

#####################################################################
# 2. Vektoren und Matrizen
#####################################################################
#vektoren werden als Spaltenvektoren betrachtet
vec1 <- c(1,3,5,7)
vec1
( vec1 <- c(1,3,5,7) )
( vec2 <- 1:4 )
( vec3 <- rep(1, times=7) ) # man wiederholt die 1 7mal
( vec4 <- rep(c(1,2), times=2) )
( vec5 <- rep(c(1,2), each=2) ) # jeder eintrag des Vektors wird 2 mal wiederholt
( vec6 <- c(c(1,2,3), vec1) ) # man stapelt Vektoren aufeinander

( mat1 <- matrix(1:4) ) #eine Matrix ohne Angabe von
#einer Zeilen- oder Spaltenanzahl wird ein Spaltenvektor
( mat2 <- matrix(c(1,2,3,4), ncol=2) )
( mat3 <- matrix(c(1,2,3,4), ncol=2, byrow=TRUE) ) # hier werden zuerst Zeilen aufgefüllt
( mat4 <- matrix(c(vec1,vec2), ncol=2) )
( mat5 <- matrix(c(1:3,c(1,2,3)), ncol=2) )
( mat6 <- matrix(NA, ncol=3, nrow=2) )

mat3[2,1] # Auswahl eines Elements
mat3[2,] # Auswahl einer Zeile
mat3[,1] # Auswahl einer Spalte
(mat6[1,2]  <- 3)
vec6[3]
```

```
vec1 * vec2 # Komponentenweises Produkt
vec1 %*% vec2 # Skalatprodukt
t(vec1) %*% vec2 # andere Schreibweise für das Skalarprodukt
vec1 %*% t(vec2)

( vec7 <- rep(1, times = 8) + 1:3 )
( vec8 <- c(1,2,3,4) + 3 )

#####################################################################
# 3. Logische Operatoren, if-Befehl, for-Schleife, Funktionen
#####################################################################
if (1==1) {"richtig"} else {"falsch"}
if (1==2) {"richtig"} else {"falsch"}
d <- 5
if (d>=0) {sqrt(d)} else {"positive Zahl übergeben"}

# if kann man auch mit mehreren Bedingungen verknüpfen
# z. B. bei einer quadratischen Gleichung ax^2+bx+c gilt,
# dass die Diskriminate über die Arten der Lösungen entscheidet:
a<- 1/4
b<- -1
c<- 1
d <- b^2-4*a*c
if (d>0) {"Die Gleichung hat zwei verschiedene reelle Lösungen"
          } else if (d==0) {"Die Gleichung hat eine doppelte reelle Lösungen"
          } else{"Die Gleichung hat zwei verschiedene komplexe Lösungen"}

n1 <- 0
for (i in 1:6) { n1 <- n1 + i }
n1

for (i in 1:6)
{
if (i == 1)
n2 <- i
else
n2 <- n2 + i
}
n2
```

```
XdurchY <- function(x,y=1)
{
if (y==0)
print("Zweiter Eintrag darf nicht 0 sein!")
else
x / y
}
# die meisten Befehle, die man benutzt, sind Funktionen;
# wichtig ist die Angabe der Funktionsvariablen:
XdurchY(28,4)
XdurchY(x=28, y=4)
XdurchY(y=4, x=28)
XdurchY(4,28)
# Fazit: indem man die Variablen im Funktionsaufruf definiert,
#kann man einerseits die Anordnung beliebig ändern,
# andererseits wird die Sache übersichtlicher
XdurchY(28,0)

#####################################################################
# 4. Zufallszahlen
#####################################################################
set.seed(42) # zur möglichen Wiederholung der "Zufallsziehung"
x <- runif(50, min=2, max=7)
y <- 14 + 3*x + rnorm(50, mean=0, sd=4)
#später wichtig: eine Ansammlung aller möglichen Verteilungsnamen
?distribution

#####################################################################
# 6. OLS-Regression
#####################################################################
ols <- lm(y ~ 1 + x)
summary(ols)

plot(x,y)
abline(ols)

u_hat <- residuals(ols)
y_hat <- predict(ols)

hist(u_hat)
```

```
#####################################################################
# 7. Graphiken
#####################################################################
plot(vec1, vec2)

pdf("plot1.pdf")

plot(x,y)
abline(ols)
dev.off()

# Trainingsaufgaben: Funktionen vecsum und scalprod

vecsum <- function(x)
{
for (i in 1:length(x))
{
if (i == 1) {zw <- x[1]} else {zw <- zw+x[i]}
}
zw
}
vecsum(vec1)
vecsum(c(1,2,3,4,5,6))

scalprod  <- function(x1,x2)
{
if (length(x1) != length(x2))
{
"Die beiden Vektoren müssen die gleiche Länge haben!"
}
else
{
for (i in 1:length(x1))
{
if (i == 1) {zw <- x1[1]*x2[1]} else {zw <- zw + x1[i]*x2[i]}
}
zw
}}
scalprod(vec1,vec1)
scalprod(c(1,2,3),c(1,2)
```

Trainingsaufgaben

- Geben Sie den Vektor `x1` ein. Dieser habe die Einträge `1,4,19`.
- Geben Sie den Vektor `x2` ein. Dieser habe die Einträge `14,-6,0.3`.
- Berechnen Sie `x3` als die Summe aus den beiden Vektoren `x1` und `x2`.
- Fügen Sie die beiden Vektoren `x1` und `x2` spaltenweise zur Matrix `M1` zusammen.
- Überprüfen Sie jeweils mit einer `if`-Abfrage, ob der Vektor `x1` genau 2 bzw. 3 Einträge hat.
- Welche Objekte befinden sich nun in Ihrem Workspace?
- Löschen Sie diese alle ohne sie explizit anzugeben mit einem einzigen Befehl (diesen Befehl müssen Sie erst suchen, z.B. mittels `?rm`).
- Erstellen Sie die Funktion $f(x,y) = \begin{cases} x-y, & \text{falls } x \geq y \\ 0, & \text{sonst} \end{cases}$ mit Hilfe des `function`-Befehls und einer `if`-Abfrage und berechnen Sie anschließend $f(5,7)$ und $f(7,5)$.
- Erstellen Sie die Funktion $g(x,y) = \begin{cases} x-y, & \text{falls } x \geq y \\ 0, & \text{sonst} \end{cases}$ mit Hilfe des `function`-Befehls und der Indikatorfunktion und berechnen Sie anschließend $g(5,7)$ und $g(7,5)$.
  Die Indikatorfunktion $I_{\{.\}}$ ist die Funktion, die den Wert 1 annimmt, falls die Bedingung in geschweiften Klammern zutrifft, und den Wert 0, falls die Bedingung nicht zutrifft.
  Hinweis: Geben Sie die Indikatorfunktion mit Hilfe eines logischen Operators an.
- Erzeugen Sie die Vektoren `x4` und `x5` mit jeweils 100 Einträgen. Dabei seien die Elemente von `x4` gleichverteilt auf $[-2;2]$ und die von `x5` seien $N(3;4)$-verteilt. Verwenden Sie den Randomseed 42.
- Plotten Sie `x4` und `x5`.
- Regressieren Sie `x5` auf `x4` und zeichnen Sie die Regressionsgerade in Ihren Plot ein.
- Erklären Sie die beiden Funktionen `vecsum` und `scalprod`, die in diesem R-Skript zu finden sind.
  Vergleichen Sie diese mit den bereits in `R` implementierten Funktionen `sum` und `crossprod`.

1. Aufgabe (0 Punkte) (*Matrizen*)

Geben Sie folgende Matrizen in `R` ein und überprüfen Sie Ihre Eingaben.

$$\mathbf{A} = \begin{pmatrix} 3 & -3 & 1 \\ -3 & 2 & 0 \\ 1 & 0 & 4 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

Führen Sie die folgenden Aufgaben mit `R` aus.

(a) Bilden Sie $\mathbf{A}^T$ und $\mathbf{B}^T$.
Hinweis: Benutzen Sie die `R`-Hilfe oder die R-intro.

(b) Geben Sie die Spalten- und Zeilenanzahl von $\mathbf{A}$ aus.

(c) Überprüfen Sie, ob die Matrizen $\mathbf{A}$ und $\mathbf{B}$ symmetrisch sind.

(d) Berechnen Sie $\mathbf{A} + \mathbf{B}$ und $\mathbf{A} - \mathbf{B}$.

(e) Berechnen Sie

i. `C1 = A*A` und

ii. `C2 = A %*% A`.

Können Sie sich die Unterschiede in den Ergebnissen erklären?

(f) Berechnen Sie $\mathbf{D} = 2 \cdot \mathbf{A}$.

(g) Berechnen Sie $\mathbf{AB}$ und $\mathbf{BA}$ und vergleichen Sie Ihre Ergebnisse.

(h) Setzen Sie $b_{32} = 0$ und wiederholen Sie die vorherige Aufgabe.

(i) Berechnen Sie das elementweise Produkt der Matrizen $\mathbf{A}$ und $\mathbf{B}$, das gegeben ist durch $a_{ij} * b_{ij}$, für alle $i = 1, \ldots, m$ und $j = 1, \ldots, n$.

(j) Bestimmen Sie die Inversen von $\mathbf{A}$ und $\mathbf{B}$.
Hinweis: Benutzen Sie die `R`-Hilfe oder lesen Sie beispielsweise Kapitel 5.7.2 in der R-intro.

2. Aufgabe (0 Punkte) (*KQ-Schätzer*)

Es liegen folgende Stichprobenwerte zur Berechnung eines einfachen Regressionsmodells $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$ vor:

$$\mathbf{X} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 8 \\ 1 & 6 \\ 1 & 2 \\ 1 & 7 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 5 \\ 1 \\ 0 \end{pmatrix}.$$

(a) Bestimmen Sie $\mathbf{X}^T\mathbf{X}$, $\mathbf{X}^T\mathbf{y}$, $\mathbf{y}^T\mathbf{y}$ und $\mathbf{y}\mathbf{y}^T$.

(b) Bestimmen Sie Dimension und Rang von $\mathbf{X}$, $\mathbf{X}^T\mathbf{X}$ und $\mathbf{X}\mathbf{X}^T$.
Hinweis: Die Funktion zur Bestimmung des Rangs einer Matrix ist im Paket `Matrix` enthalten, das zwar nicht mehr installiert werden muss, aber noch über `library(Matrix)` geladen werden muss.

(c) Bestimmen Sie die Inverse von $\mathbf{X}^T\mathbf{X}$.

(d) Berechnen Sie den KQ-Schätzer für $\boldsymbol{\beta}$.

(e) Berechnen Sie die Fehlerquadratsumme, einen Schätzer der Fehlervarianz und R^2.

(f) Schreiben Sie eine Funktion `BetaHat`, die den OLS-Schätzer für den Vektor $\mathbf{y}$ und die Matrix $\mathbf{X}$ berechnet. Achten Sie dabei darauf, dass die Dimensionen von $\mathbf{X}$ und $\mathbf{y}$ zusammenpassen.
Hinweis: Zur Orientierung können Sie die Funktion `scalprod` aus diesem R-Skript verwenden.

3. Aufgabe (0 Punkte) (*Regression*)

Im Folgenden wird das Modell

$$\log(wage) = \beta_1 + \beta_2 educ + u$$

untersucht.

(a) Laden Sie den Datensatz wage2.txt in `R`. Dabei handelt es sich um einen Auszug aus dem Datensatz wage2.wf1 aus Wooldridge (2009) im .txt-Format.
Wie viele Beobachtungen liegen für *wage* vor?

(b) Schätzen Sie obiges Modell mit Hilfe der Funktion `lm` und interpretieren Sie den `summary`-Output.
Achten Sie darauf, dass Sie die Variable *wage* im Logarithmus verwenden.

(c) Speichern Sie die Residuen und die gefitteten Werte und geben Sie ein Residuen-Histogramm aus.

(d) Erstellen Sie einen *educ*-log(*wage*)-Plot und zeichnen Sie die OLS-Regressionsgerade ein.
Speichern Sie Ihre Graphik als pdf.