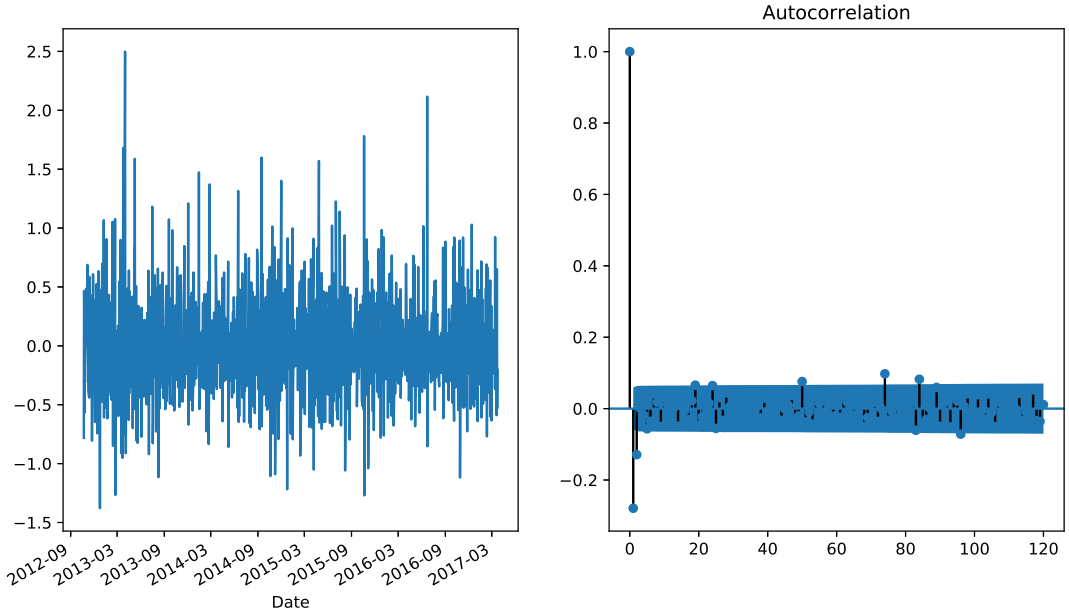
Musterlösungen zu Serie 13

Lösung 13.1

$$\begin{aligned}\gamma(i,j) &= \mathrm{E}\left((X_i - \mu(i))(X_j - \mu(j))\right) \\ &= \mathrm{E}\left(X_i X_j\right) - \mu(i)X_j - \mu(j)X_i + \mu(i)\mu(j)\right) \\ &= \mathrm{E}(X_i X_j) - \mu(i)\mu(j) - \mu(j)\mu(i) + \mu(i)\mu(j) = \mathrm{E}(X_i X_j) - \mu(i)\mu(j).\end{aligned}$$

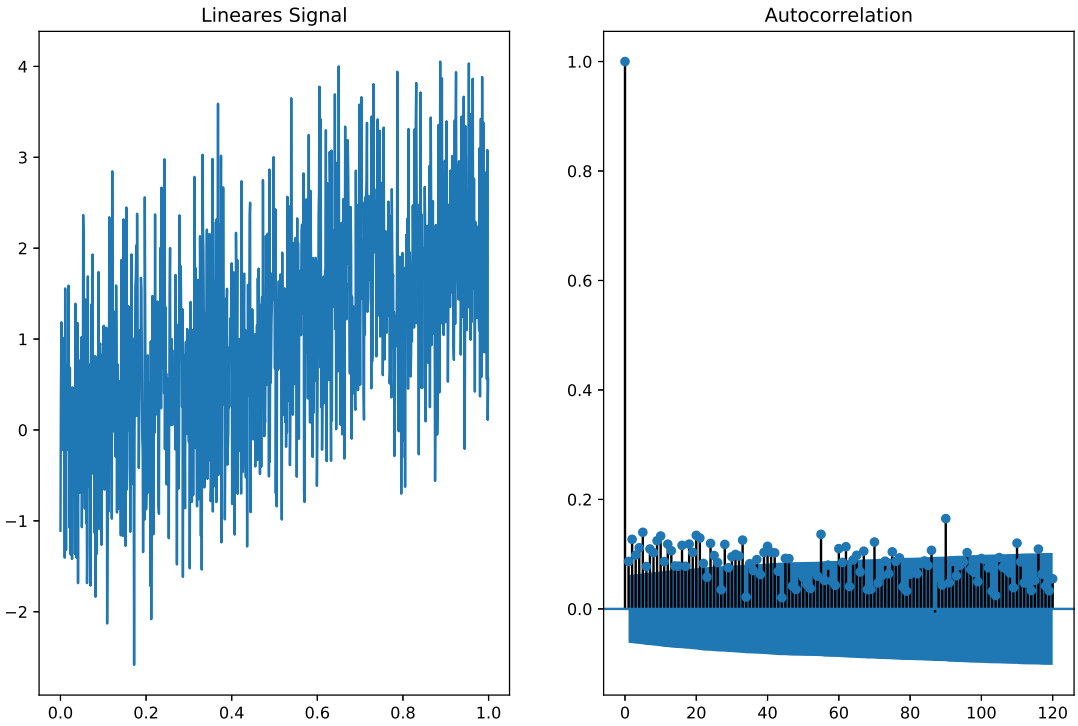
Lösung 13.2

a)
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas import DataFrame
from statsmodels.graphics.tsaplots import plot_acf
Tesla = pd.read_csv("/Users/mirkobirbaumer/Dropbox (CC-IIMSN)/Statistics/Th

Tesla["Date"] = pd.DatetimeIndex(Tesla["Date"])
Tesla.set_index("Date", inplace=True)

Tesla["log_volume"] = np.log(Tesla["Volume"])
Tesla["log_return"] = Tesla["log_volume"] - Tesla["log_volume"].shift(1)
fig, (ax1, ax2) = plt.subplots(ncols=2)
Tesla["log_return"].plot(ax=ax1)

plot_acf(DataFrame(Tesla["log_return"]).dropna(), lags=120, ax=ax2)
```



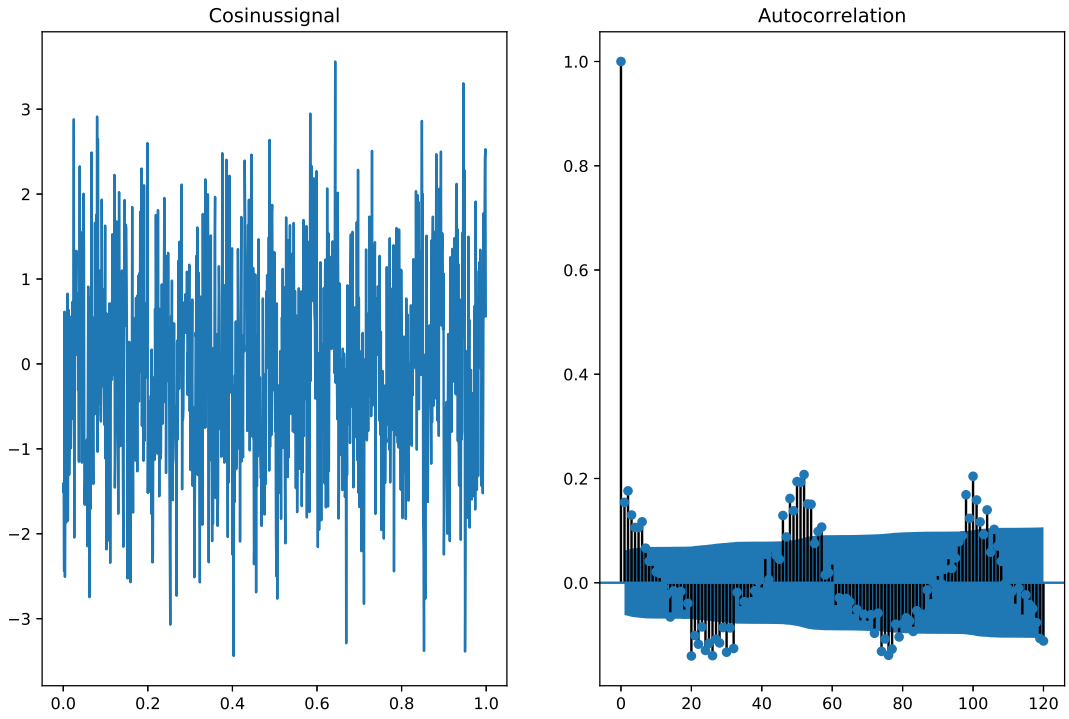
b)

```
from statsmodels.graphics.tsaplots import plot_acf
from pandas import DataFrame
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0, 1, 1000, endpoint=False)
noise = np.random.normal(size=1000)
signal = 2*t + noise
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title('Lineares Signal')
ax1.plot(t, signal)
plot_acf(DataFrame(signal + noise).dropna(), lags=120, ax=ax2)
plt.show()
```



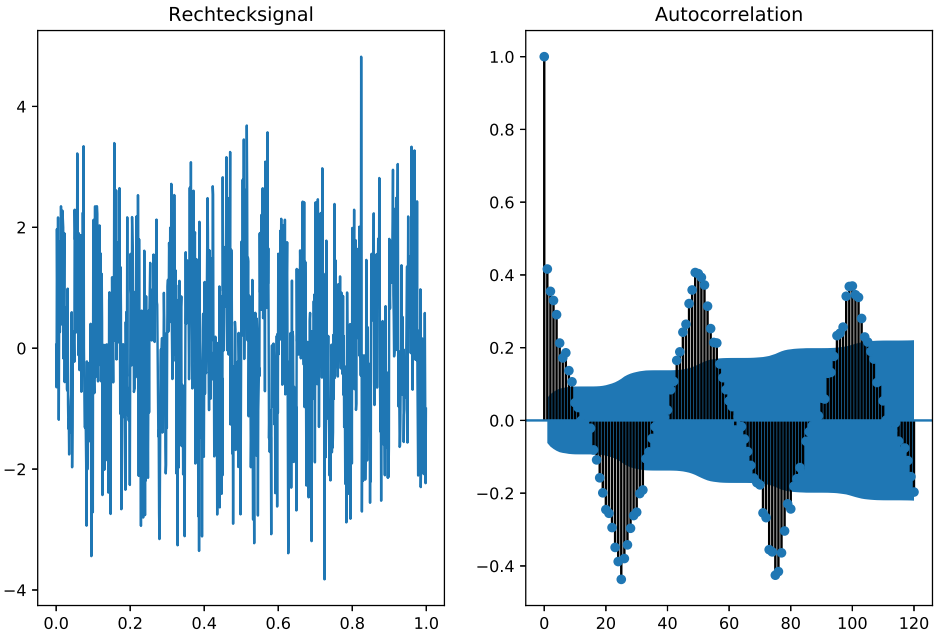
c)

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0, 1, 1000, endpoint=False)
noise = np.random.normal(size=1000)
signal = signal.sawtooth(2 * np.pi * 20 * t)
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title('Cosinussignal')
ax1.plot(t, signal + noise)
plot_acf(DataFrame(signal + noise).dropna(), lags=120, ax=ax2)
plt.show()
```



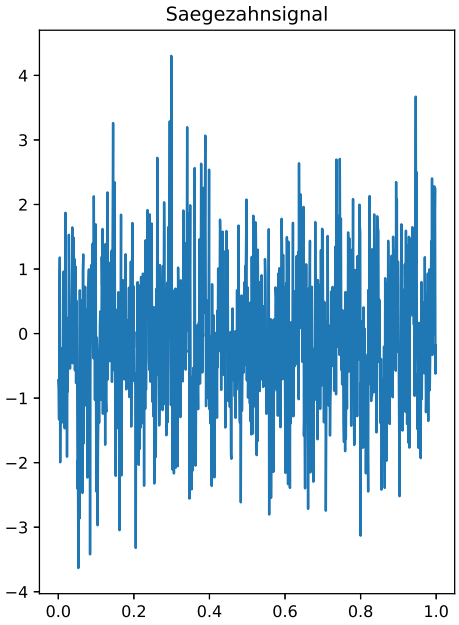
d)

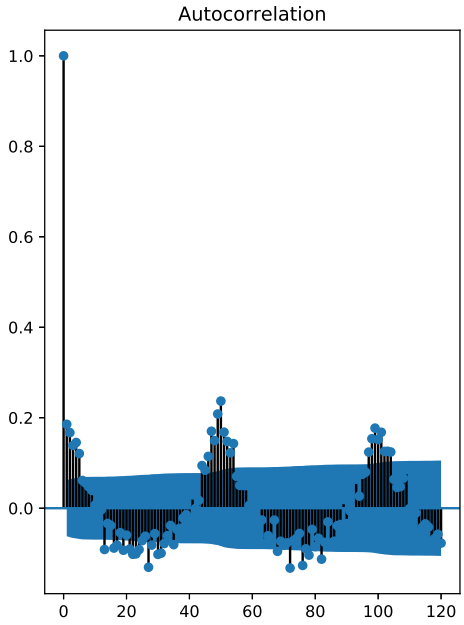
```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0, 1, 1000, endpoint=False)
noise = np.random.normal(size=1000)
signal = signal.square(2 * np.pi * 20 * t)
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title('Rechtecksignal')
ax1.plot(t, signal + noise)
plot_acf(DataFrame(signal + noise).dropna(), lags=120, ax=ax2)
plt.show()
```



e)

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0, 1, 1000, endpoint=False)
noise = np.random.normal(size=1000)
signal = signal.sawtooth(2 * np.pi * 20 * t)
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title('Saegezahnsignal')
ax1.plot(t, signal + noise)
plot_acf(DataFrame(signal + noise).dropna(), lags=120, ax=ax2)
plt.show()
```





Lösung 13.3

a) Es gilt $\mathrm{E}[X_i] = \frac{3}{2} - i$

$$\begin{aligned}
\gamma(i,j) &= \mathrm{E}[(T + (1-i) - \frac{3}{2} + i) \cdot ((T + (1-j)) - \frac{3}{2} + j)] \\
&= \mathrm{E}[(T - \frac{1}{2}) \cdot (T - \frac{1}{2})] \\
&= \mathrm{E}[T^2] + \mathrm{E}[T] + \frac{1}{4} \\
&= \frac{1}{3} + \frac{1}{2} + \frac{1}{4} \\
&= \frac{13}{12}
\end{aligned}$$

Obwohl die Autokorrelationskoeffizienten den konstanten Wert 0 für denselben (beliebigen) lag haben, ist der Scharmittelwert von der Zeit i abhängig. Der Prozess ist folglich nicht-stationär.

b) Dann ist der Scharmittelwert

$$\mu(n) = \mathrm{E}[X_n] = \mathrm{E}[A^n] = \int_0^1 a^n \,\mathrm{d}a = \frac{1}{n+1}\,,$$

und die Autokorrelationsfunktion

$$\begin{aligned}
\gamma(n,m) &= \mathrm{E}[(X_n - \frac{1}{n+1})(X_m - \frac{1}{m+1})] \\
&= \mathrm{E}[A^{n+m}] - \frac{1}{n+1}\,\mathrm{E}[A^m] - \frac{1}{m+1}\,\mathrm{E}[A^n] + \frac{1}{(n+1)\cdot(m+1)} \\
&= \int_0^1 a^{n+m}\,\mathrm{d}a - \frac{1}{(n+1)\cdot(m+1)} \\
&= \frac{1}{n+m+1} - \frac{1}{(n+1)\cdot(m+1)}\,.
\end{aligned}$$

Da sowohl $\mu(n)$ und $\gamma(n,m)$ von den diskreten Zeitvariablen n und m abhängen, handelt es sich bei X_n **nicht** um einen stationären Prozess.

Lösung 13.4

a) Da die W_is Zufallsvariablen mit Erwartungswert null sind, folgt daraus, dass eine Linearkombination von diesen Zufallsvariabeln ebenfalls den Mittelwert null hat. Daher gilt: $\mu(i) = 0$.

b) Wir berechnen die Autokovarinanz für den lag $h = 0$, d.h., $i = j$. Benützt man die Tatsache, dass $\mathrm{E}(X_i X_j) = 0$ falls $i \neq j$, so erhalten wir

$$\begin{aligned}
\gamma(i,i) &= \mathrm{E}\left((W_{i-1} + 2W_i + W_{i+1})(W_{i-1} + 2W_i + W_{i+1})\right) \\
&= \mathrm{E}(W_{i-1}^2) + 4\mathrm{E}(W_i^2) + \mathrm{E}(W_{i+1}^2) = \sigma^2 + 4\sigma^2 + \sigma^2 = 6\sigma^2.
\end{aligned}$$

Analog finden wir für $j = i + 1$

$$\begin{aligned}
\gamma(i,i+1) &= \mathrm{E}\left((W_{i-1} + 2W_i + W_{i+1})(W_i + 2W_{i+1} + W_{i+2})\right) \\
&= 2\mathrm{E}(W_i^2) + 2\mathrm{E}(W_{i+1}^2) = 2\sigma^2 + 2\sigma^2 = 4\sigma^2.
\end{aligned}$$

Schliesslich erhalten wir für $j = i + 2$

$$\begin{aligned}
\gamma(i,i+2) &= \mathrm{E}\left((W_{i-1} + 2W_i + W_{i+1})(W_{i+1} + 2W_{i+2} + W_{i+3})\right) \\
&= \mathrm{E}(W_{i+1}^2) = \sigma^2.
\end{aligned}$$

Zusammengefasst ergibt sich:

$$\gamma(i,j) = \begin{cases} 6\sigma^2 & \text{falls } j = i \\ 4\sigma^2 & \text{falls } j = i+1 \\ \sigma^2 & \text{falls } j = i+2 \\ 0 & \text{ansonsten.} \end{cases}$$

Die Autokorrelation erhält man durch $\rho(i,j) = \gamma(i,j)/\gamma(i,i)$, da die Autokovarianz bloss vom Lag der Zeiten $h = i - j$ abhängt. Daraus ergibt sich

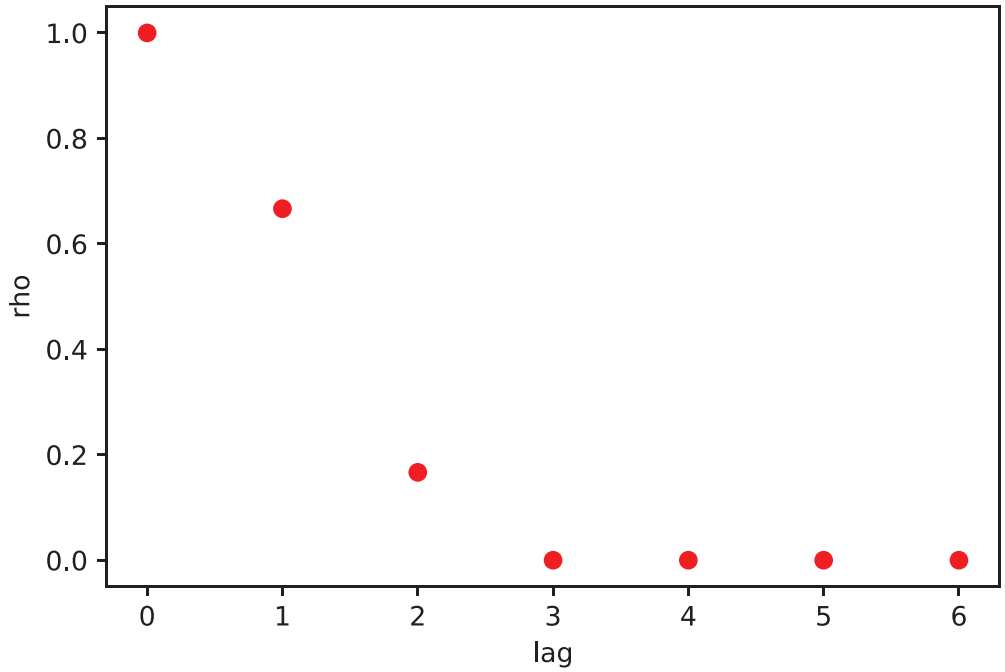
$$\rho(i,j) = \begin{cases} 1 & \text{if } j = i \\ \frac{2}{3} & \text{if } j = i+1 \\ \frac{1}{6} & \text{if } j = i+2 \\ 0 & \text{ansonsten.} \end{cases}$$

c) Die Autokorrelation als eine Funktion von lag h ist dann

$$\rho(h) = \begin{cases} 1 & \text{falls } h = 0 \\ \frac{2}{3} & \text{falls } h = 1 \\ \frac{1}{6} & \text{falls } h = 2 \\ 0 & \text{ansonsten.} \end{cases}$$

Wir können die Autokorrelationsfunktion folgendermassen in **Python** darstellen

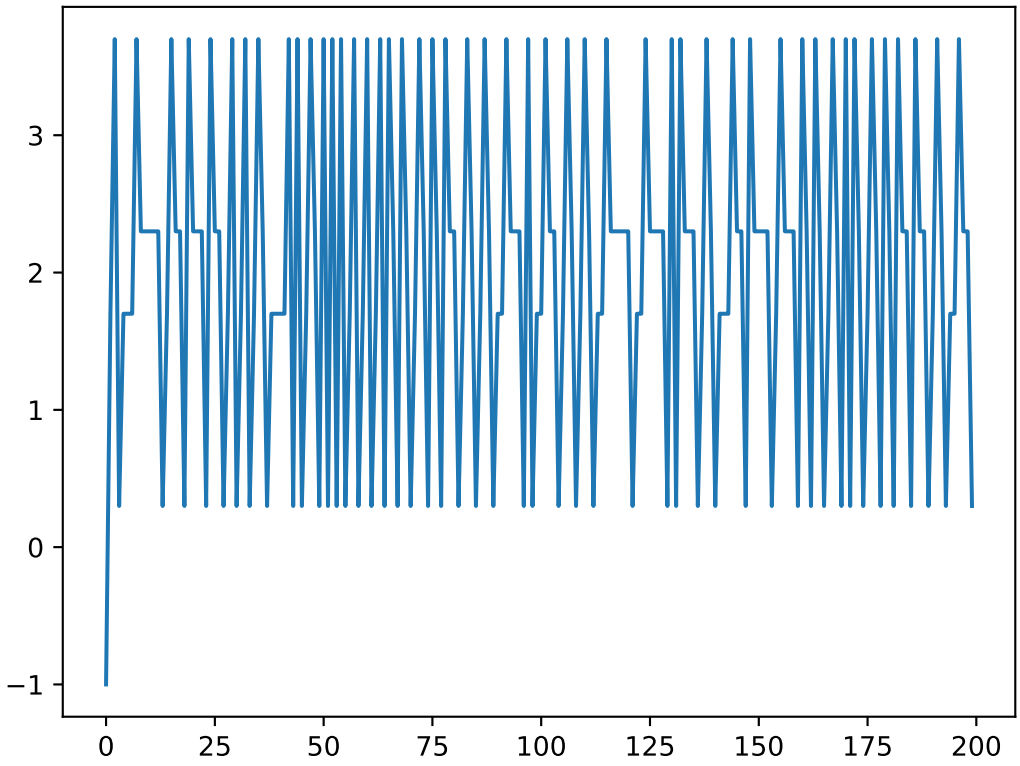
```
import numpy as np
import matplotlib.pyplot as plt

h = [(0, 1, 2, 3, 4, 5, 6)]
rho = [(1, 2/3, 1/6, 0, 0, 0, 0)]
plt.plot(h, rho, 'ro')
plt.xlabel("lag")
plt.ylabel("rho")
plt.show()
```



Lösung 13.5

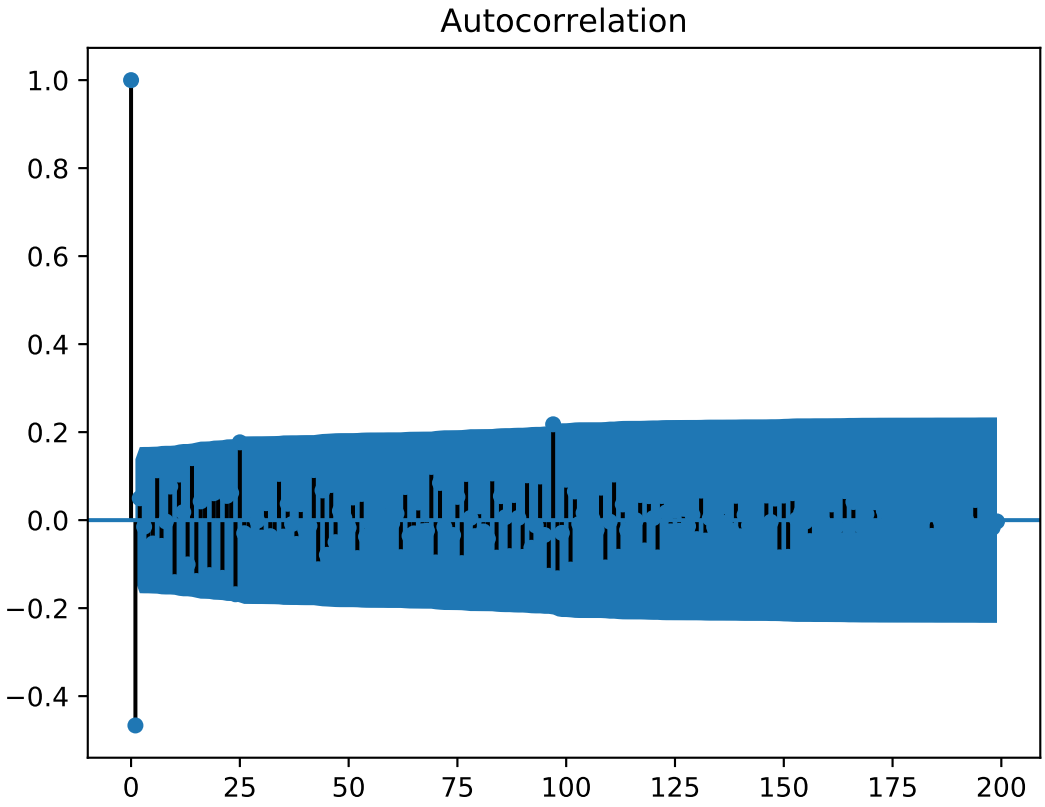
a) (zu R)

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf

# Create the process D_k
D = 2*(np.random.binomial(size=200, n=1, p= 0.5)-0.5)
# Create the process X_k
X=np.zeros(200)
X[0]=-1

for k in range(1,200):
    X[k] = 2 + D[k] - 0.7*D[k-1]
plt.plot(X)
```



b) (zu R)

```
plot_acf(X)
```



c) Für den Mittelwert berechnen wir:

$$\mu(i) = \mathrm{E}(X_i) = a + \mathrm{E}(D_k) + b\mathrm{E}(D_{k-1}) = a.$$

Wenn wir die Autokovarianz berechnen, dann nur für Paare X_k und X_j, für welche wir berücksichtigt haben, dass sie D_ks mit unterschiedlichen Indizes enthalten, da das Werfen der Münze unabhängig ist. Somit müssen wir berücksichtigen, dass $k = j$ und $k = j + 1$. Für alle anderen Fälle gilt, dass die Autokovarianz null ist.

$$\begin{aligned}\gamma(k,k) &= \mathrm{E}\left((D_k + bD_{k-1})(D_k + bD_{k-1})\right) \\ &= \mathrm{E}\left(D_k^2 + 2bD_{k-1}D_k + b^2D_{k-1}^2\right) \\ &= 1 + b^2\end{aligned}$$

Die letzte Gleichung folgt aus der Beobachtung, dass $\mathrm{E}(D_kD_{k-1}) = 0$ und $\mathrm{E}(D_k^2) =$

1. Nun berechnen wir

$$\begin{aligned}\gamma(k,k+1) &= \mathrm{E}\left((D_k + bD_{k-1})(D_{k+1} + bD_k)\right) \\ &= \mathrm{E}\left(D_k D_{k-1} + bD_k D_{k+1} + bD_{k-1}D_k + bD_k^2\right) \\ &= b\end{aligned}$$

In diesem Fall wurden alle Terme, deren Produkt einen Erwartungswert von null haben, nicht berechnet. Schliesslich erhalten wir:

$$\rho(k,j) = \begin{cases} 1 & \text{if } j = k \\ \frac{b}{1+b^2} & \text{falls } j = k+1 \\ 0 & \text{ansonsten.} \end{cases}$$

Wir haben $\frac{-0.7}{1+0.49} = -0.4697987$. Andererseits finden wir für $\hat{\rho(1)}$:

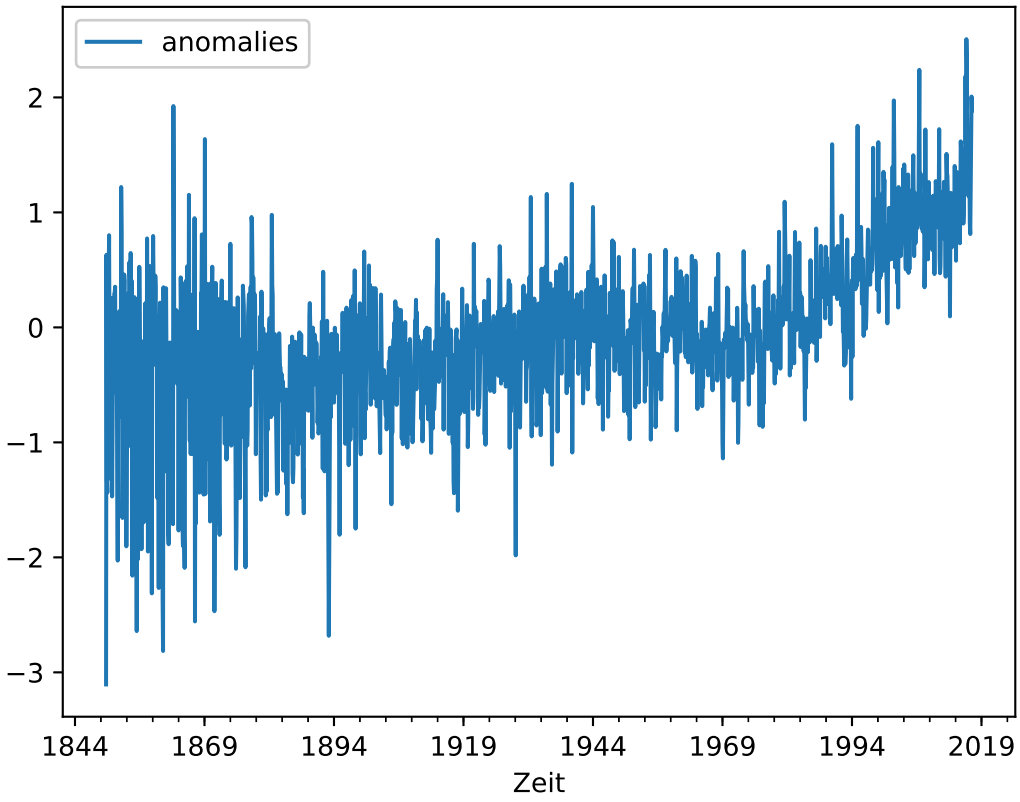
```
D = 2*(np.random.binomial(size=200, n=1, p= 0.5)-0.5)

X=np.zeros(200)
X[0]=-1

for k in range(1,200):
    X[k] = 2 + D[k] - 0.7*D[k-1]
acf(X)[0]
print(acf(X)[1])

## Error in running command /usr/bin/python3
```

d) Ja. Der Mittelwert ist konstant und die Autokorrelationsfolge hängt nur vom Lag des Argumentes ab.

Lösung 13.6

a) (zu **R**)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
global_temp = pd.read_csv("../global_temp.csv")
global_temp["Zeit"] = pd.DatetimeIndex(global_temp["Zeit"])
global_temp.set_index("Zeit", inplace=True)
print(global_temp.head())
global_temp.plot()
```



b) (zu **R**)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
global_temp = pd.read_csv("../global_temp.csv")
global_temp["Zeit"] = pd.DatetimeIndex(global_temp["Zeit"])
global_temp.set_index("Zeit", inplace=True)
seasonal_decompose(global_temp["anomalies"], model="additive", freq=10).plo
```

Die Zerlegung ergibt eine Trendkomponente mit einem signifikanten Zuwachs zwischen den 70er Jahren und jetzt. Im 18. Jahrhundert scheinen die Daten eine grössere Varianz gehabt zu haben. Dies könnte mit dem schlechten Equipment zur damaligen Zeit zu tun haben.

c) (zu **R**)

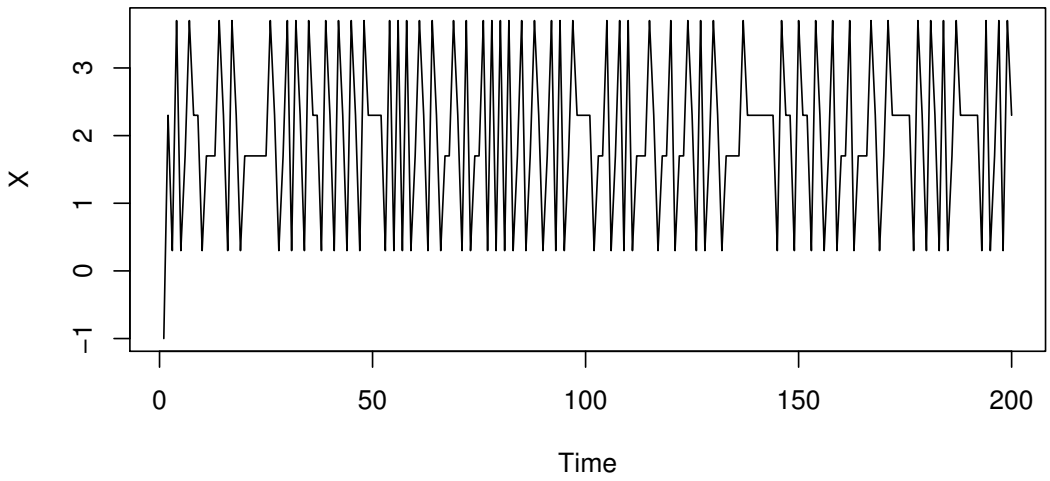
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf
global_temp = pd.read_csv(".../global_temp.csv")
global_temp["Zeit"] = pd.DatetimeIndex(global_temp["Zeit"])
global_temp.set_index("Zeit", inplace=True)
remainder = seasonal_decompose(global_temp["anomalies"], model="additive",
plot_acf(remainder, lags=21)
```

R-Code

Aufgabe 13.5

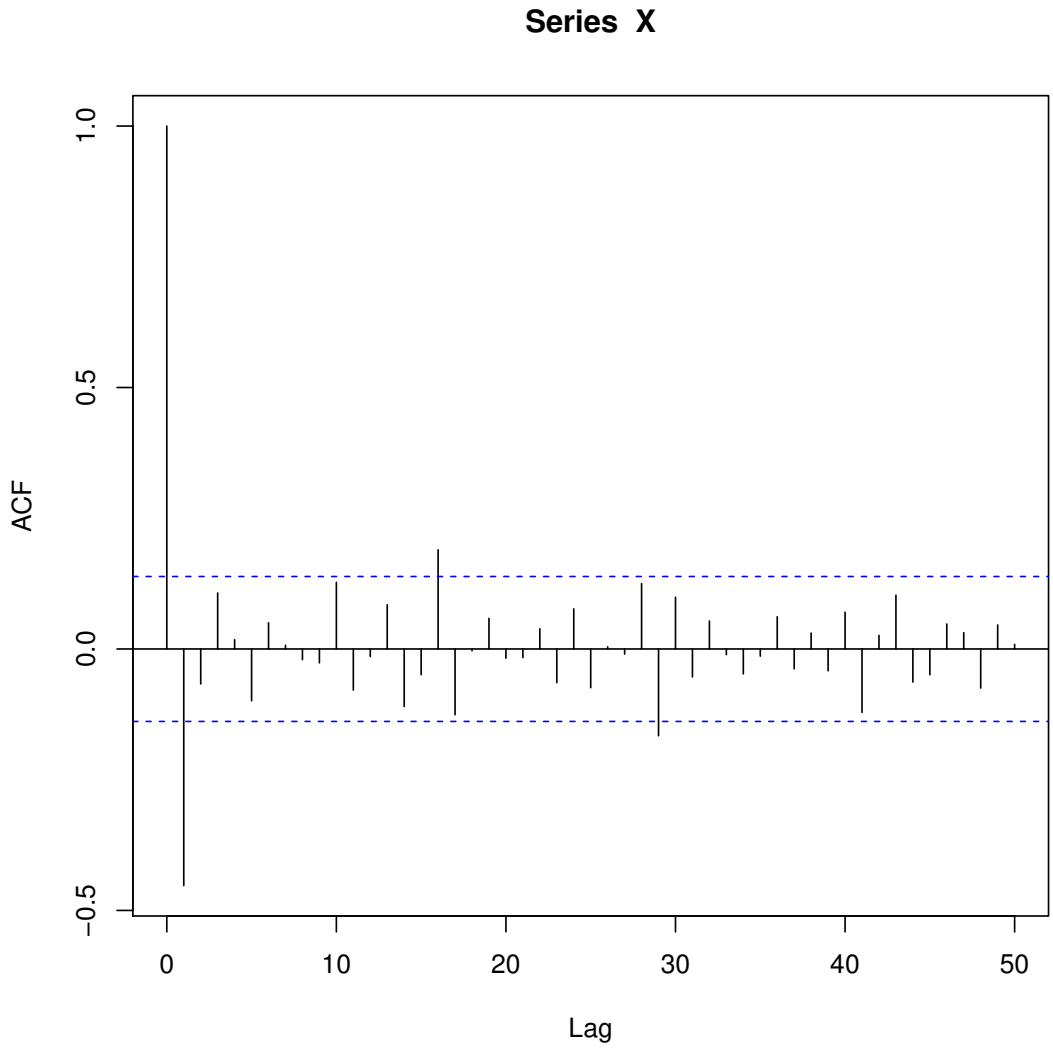
a) (zu **Python**)

```r
# Create the process Dk
D = 2 * (rbinom(200, size = 1, p = 0.5) - 0.5)

# Create the process Xk
X = rep(-1, 1, 200)
for (k in 2:200) {
    X[k] = 2 + D[k] - 0.7 * D[k - 1]
}
X = ts(X)

plot(X)
```



b) (zu **Python**)

```r
# aggregate the data
plot(acf(X, lag.max = 50))
```

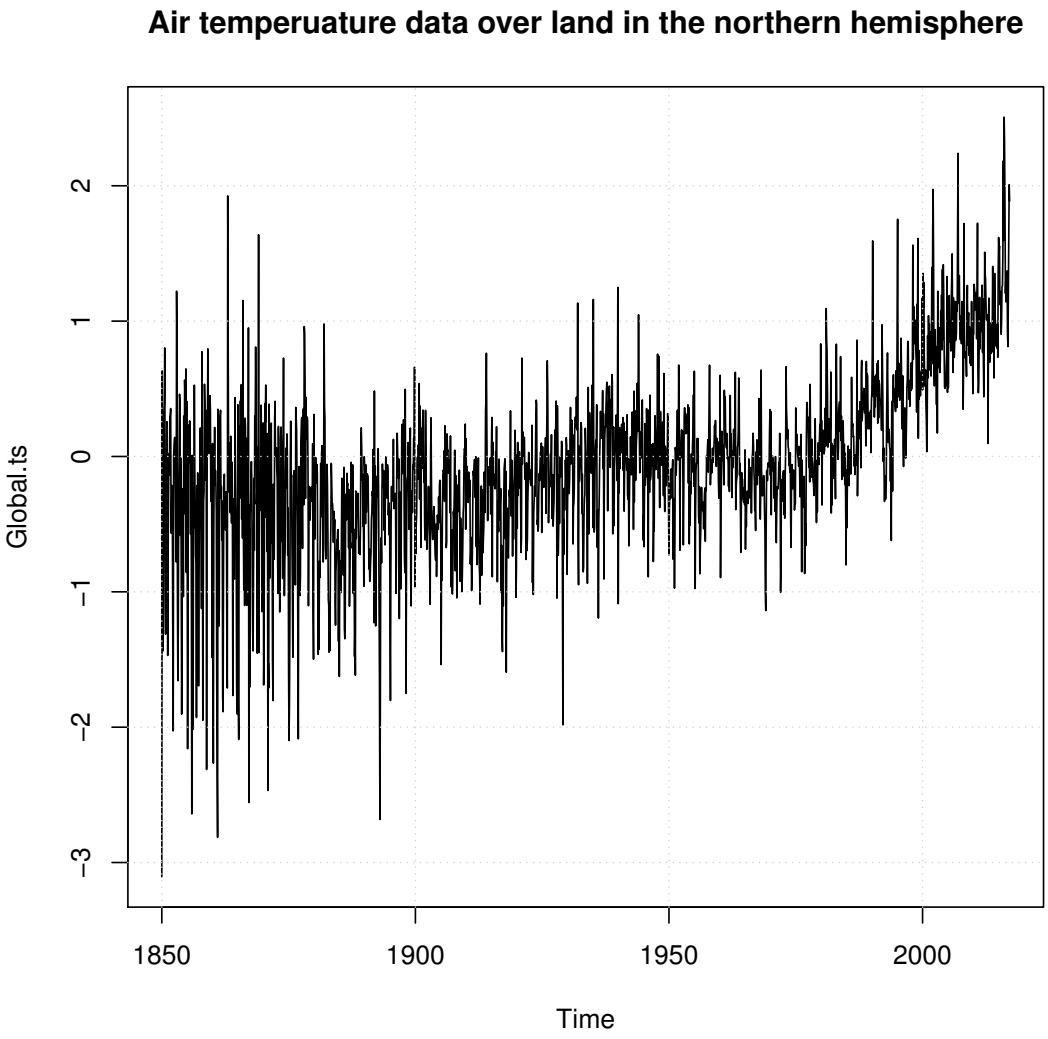


c)
```
round(acf(X, plot = F)$acf[2], 2)

## [1] -0.45

round(-0.7/(1 + 0.49), 2)

## [1] -0.47
```

Aufgabe 13.6

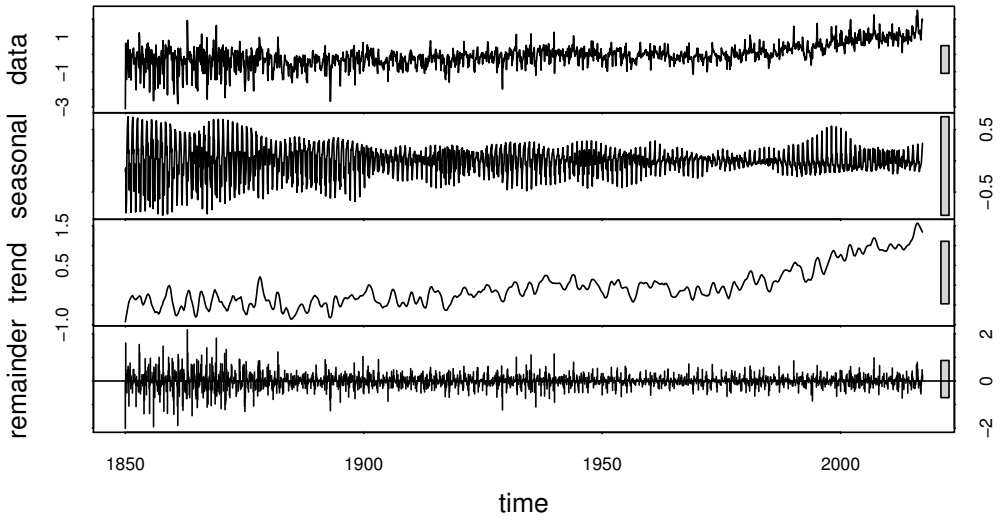
a) (zu **Python**)
```
load("../../../Themen/Time_Series_mathematical_models/Uebungen_de/Daten/Glo
plot(Global.ts, main = "Air temperuature data over land in the northern hem
```

```
grid()
```



b) (zu **Python**)

```
res.stl = stl(Global.ts, s.window = 10)
plot(res.stl)
```



c) (zu **Python**)

```
acf(res.stl$time.series[, 3], lag.max = 100)
```