Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Signale, Systeme und Sensoren

Aufbau eines einfachen Spracherkenners

Tobias Schoch, Luca Strattmann

Konstanz, 26. Mai 2019

Zusammenfassung (Abstract)

Thema:	Aufbau eines einfachen Spracherkenners	
Autoren:	Tobias Schoch	tobias.schoch@htwg-konstanz.de
	Luca Strattmann	luca.strattmann@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz	mfranz@htwg-konstanz.de
	Jürgen Keppler	juergen.keppler@htwg-konstanz.de
	Christoph Kaiser	ch241kai@htwg-konstanz.de

Zusammenfassung etwa 100 Worte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1

Einleitung

[?] [?]

2

Versuch 1

2.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Fragestellung:
In dem ersten Teil des Versuchs *'Aufbau eines einfachen Spracherkenners'* werden wir einen Spracherkenner bauen. Zudem erweitern wir den Spracherkenner mit einer Triggerfunktion. Mit der Aufnahme bestimmen wir dessen Amplitudenspektrum. Im Anschluss werden die Methode des Windowing anwenden.
So werden wir eine Testaufnahme machen um diese anschließend auszuwerten.
Auf die entstandenen Numpy-Dateien wenden wir Windowing und das Amplitudenspektrum an.

Messprinzip:
Im ersten Versuch starten wir mit einem Pythonskript. Das Mikrofon wird mit einem Klinkenstecker direkt an die Soundkarte des Computers verbunden. Auf den Computern im Labor, ist das Paket PyAudio bereits in den IDE's integriert.
Mit dem geschriebenen Pythonskript können wir nun akustische Signale aufnehmen. Über das Objekt *audiorecorder* haben wir Zugriff auf die Aufnahmefunktion der Soundkarte.
Das Signal speichern wir anschließend mittels *numpy.save()*. Im Anschluss sollen wir eine beliebige Spracheingabe aufnehmen und diese in einem Diagramm darstellen.

Im Anschluss sollen wir das Aufnahmeprogramm um eine Triggerfunktion erweitern, welche die Aufnahme erst ab einem gewissen Lautstärkepegel starten lässt.
So können wir sicher stellen, dass alle Aufnahmen den selben Startpunkt besitzen.

Das Signal soll eine Dauer von einer Sekunde haben und die fehlenden Samples mit Nullen augefüllt werden.

Mit dem Code Aus dem dritten Versuch können wir mit der Aufnahme das Amplitudenspektrum bestimmen. Dies stellen wir grapisch dar. Danach implementieren die Methode des Windowing.
Diese werden wir jeweils in einer Länge von 512 Samples darstellen. Die einzelnen Windows werden wir mit der Gaußschen Fensterfunktion multiplizieren die eine Fensterbreite von der Standartabweichung 4 hat.
Den ersten Versuch werden wir mit dem Amplitudenspektrum erneut überprüfen und so das Spektrum aus der letzten Aufgabe auf Korrektheit überprüfen.

Aufbau:
Das Mikrofon wird durch einen Klinkenstecker direkt an die Soundkarte des Computers verbunden. Durch ein Pythonskript können wir nun Sprachaufnahmen machen.

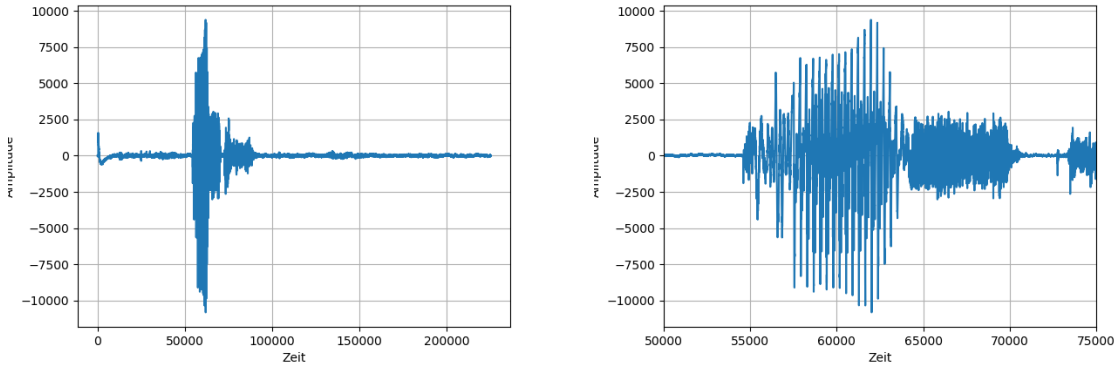
Abbildung 2.1: Aufnahme mit Triggerfunktion des Wortes rechts

Messmittel:

- Ein Mikrofon
- Ein Computer mit einer Python IDE

2.2 Messwerte

Mit einem Mikrofon das an den Computer angeschlossen ist, sehen wir die mit dem Python-skript aufgenommene Sprachaufnahme des Wortes *'Test'*, welche mit Python visualisiert wurde.
In der linken Darstellung ist die gesamte Dauer des empfangenen Signals, welche sich durch einen einfachen Plot des Signals darstellen lässt.
Im rechten Bild ist eine verkürzte Darstellung der Sprachaufnahme um das Signal besser zu erkennen welches durch *plt.xlim(0, 35000)* auf eine Frequenz von 35000 limitiert wird.



(a) Sprachaufnahme des Wortes Test (b) Dieselbe Sprachaufnahme mit kurzer Zeitachse

Abbildung 2.2: Die aufgenommene Sprachaufnahme visualisiert mit Python

Die Aufnahme wurde durch die Funktion *p.open()* gestartet.
Im Gegensatz zum zweiten Teil der Aufgabe wird hier bereits durch das Starten des Skriptes aufgenommen.

Im folgenden sieht man die Sprachaufnahme des Wortes ’Rechts’ mit der Triggerfunktion. Die Aufnahme läuft bereits durch das Starten des Programmes.
Die Stelle bei der die Überschreitung des Schwellenwertes geschieht wird gespeichert und beginnt ab dieser Stelle.
Durch die Triggerfunktion startet die Aufnahme dadurch erst ab dem Schwellenwert.

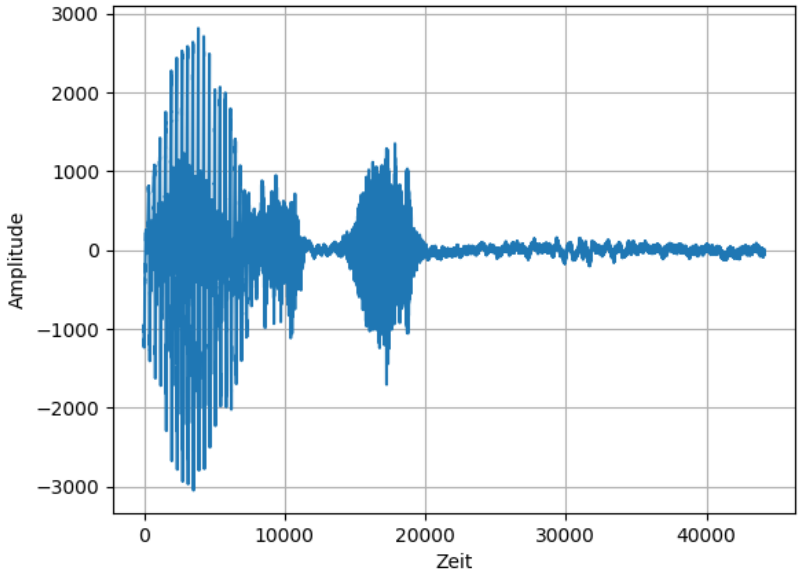
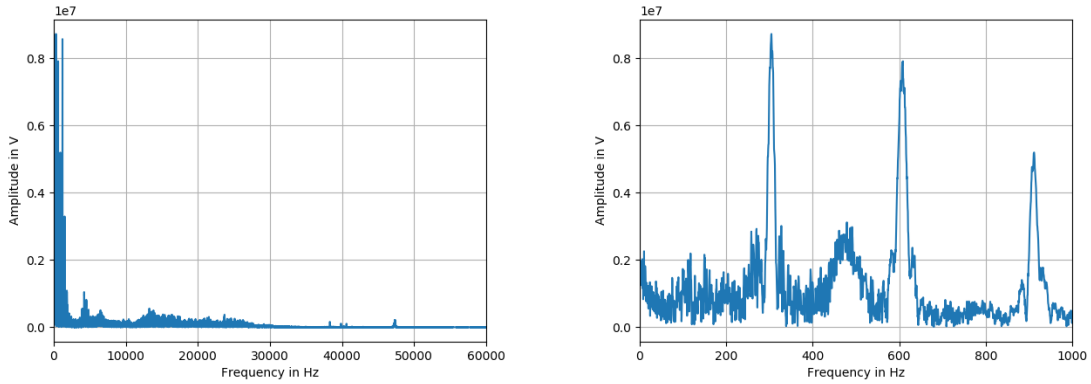


Abbildung 2.3: Aufnahme mit Triggerfunktion des Wortes rechts

2.3 Auswertung

Auf der linken Seite ist das Amplitudenspektrum und auf der rechten Seite das Amplitudenspektrum bis zur Frequenz von 1000.
Diese wurden mit der Formel berechnet. Mit der Anwendung der Formel auf jedes eingegangene Signal erhalten wir den folgenden Plot:



(a) Amplitudenspektrum der Sprachaufnahme (b) Amplitudenspektrum mit geringerer Frequenz

Abbildung 2.4: Das Amplitudenspektrum mit der dazugehörigen Frequenz

Im folgenden sieht man eine Gaußsche Fensterfunktion mit der Fensterbreite von 4 Standardabweichungen.

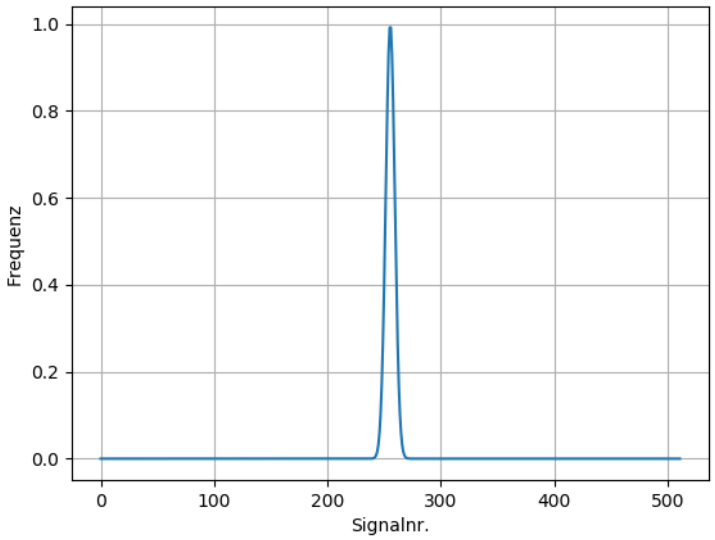


Abbildung 2.5: Gausssche Fensterfunktion mit Standardabweichung 4

Im Anschluss zerlegt man das Signal in Abschnitte mit der von Länge 512 Samples, die sich jeweils zur Hälfte überlappen sollen. Das Signal wird mit dem entsprechenden Signal der Gaußschen Fensterfunktion multipliziert und daraufhin eine lokale Fouriertransformation durchgeführt. Zu guter Letzt werden die Fouriertransformierten über alle Fenster gemittelt.

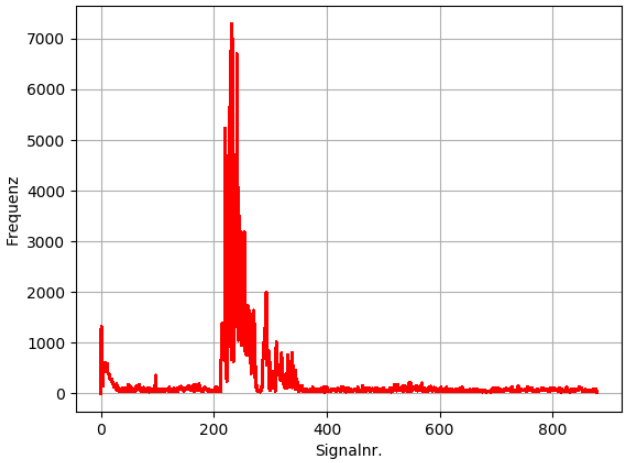


Abbildung 2.6: Aufnahme mit Triggerfunktion des Wortes rechts

Das Windowing wird auf das Signal angewandt. Nachdem zum Beispiel bei dem Wort 'Hoch' über 170 Windows ausgegeben wurden als Plot, sieht der Explorer folgendermaßen aus.

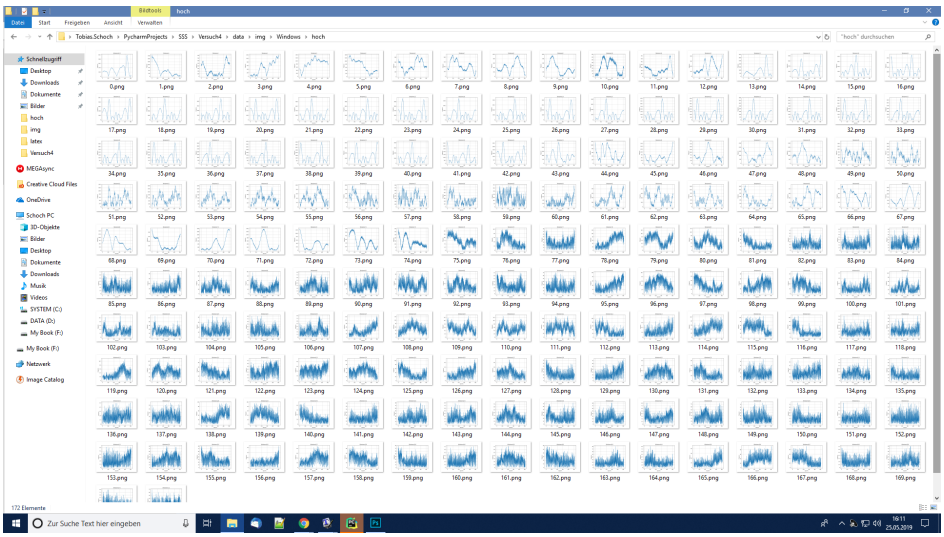


Abbildung 2.7: Aufnahme mit Triggerfunktion des Wortes rechts

2.4 Interpretation

Bei der Betrachtung der Sprachaufnahme sehen wir die übliche Amplituden und Frequenzen bei der Aussprache des Wortes 'Test'. In der Vergrößerung sind nochmals besser die Amplitudenausschläge zu sehen.

In der Abbildung 2.3 sieht man die Sprachaufnahme des Wortes 'Rechts'. Die Aufnahme wurde mit einem Pythonskript gemacht, dass eine Triggerfunktion implementiert hat. Ab einem bestimmten Schwellenwert beginnt die Aufnahme. Hier ist es schön zu sehen, dass die Aufnahme nicht mit Start des Skriptes beginnt, sondern erst wenn gesprochen wird.

Auf das Signal wurde daraufhin das Amplitudenspektrum berechnet. Durch dieses erfährt man die Frequenzen die in einem Wort stecken. Der größte Ausschlag bei der Frequenz mit eienr Amplitude von über 0.8 ist die maximalste Amplitude. Die Frequenz liegt bei knapp über 300Hz. Die x Achse wurde auf die Hälfte gekürzt, da bei ein Amplitudenspektrum sich nach der Hälfte wiederholt.

Wenn man diese mit dem Gaußschen Fenster multipliziert, erhält man die gemittelten fouriertransfomierten Fenster. Desto höher der absolute Ausschlag, desto höher ist die Amplitude im Window. Durch den senkrechten Ausschnitte sind steile Übergänge erzeugt worden, die im ursprünglichen Signal nicht vorhanden sind. Steile Übergänge erzeugen jedoch viele hohe Frequenzen im Spektrum.

3

Versuch 2

3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im zweiten Versuch müssen wir vier verschiedene Befehle jeweils fünf mal aufnehmen.
Die Befehle die wir aufnehmen lauten:

- Links
- Rechts
- Hoch
- Tief

Anhand der aufgenommenen Numpy Funktionen berechnen wir die Spektren mit der Windowing Methode. Daraufhin berechnen wir noch die eigentlichen Referenzspektren.
Im Anschluss berechnen wir noch den Korrelationskoeffizienten nach Bravais-Pearson mit dem wir zwei verschiedene Spektren miteinander vergleichen können.
Dafür brauchen wir vom selben und einem anderen Sprecher erneut Sprachaufnahmen.
Mit diesen testen wir die Routine an den Referenzspektren:
Beim Vergleich identischer Spektren sollte die Korrelation 1 sein, bei verschiedenen Spektren nahe an 0.
Zudem sollen wir eine Fehler- und eine Detektionsrate angeben.

3.2 Messwerte

Die sämtlichen aufgenommenen Sprachaufnahmen in Form von Numpy-Dateien werden wir mit der Windowing Methode in Spektren berechnen. Dabei wird erstmal das Spektrum mit der Gaußschen Fensterfunktion berechnet und multipliziert mit den einzelnen Windows. Danach wird das Fenster absolut fouriertransformiert. Daraus wiederrum wird der Durchschnitt berechnet.

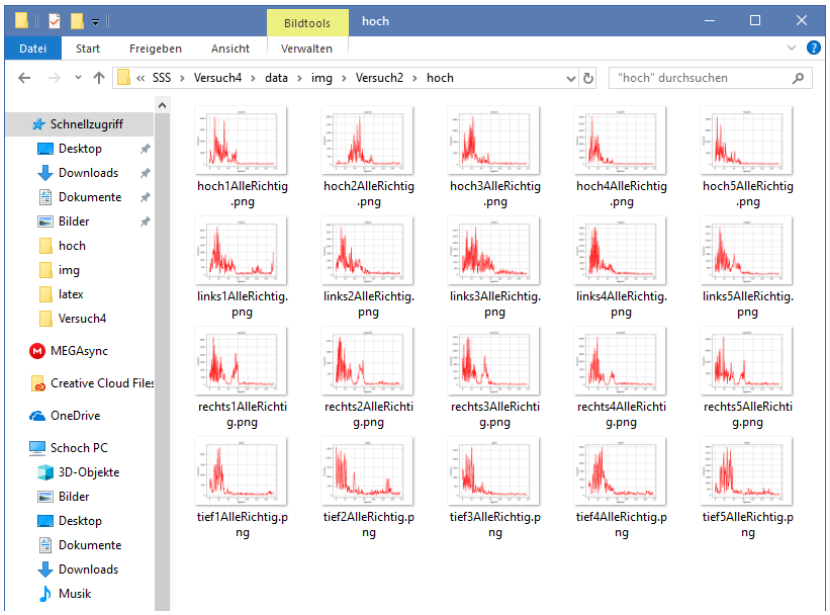
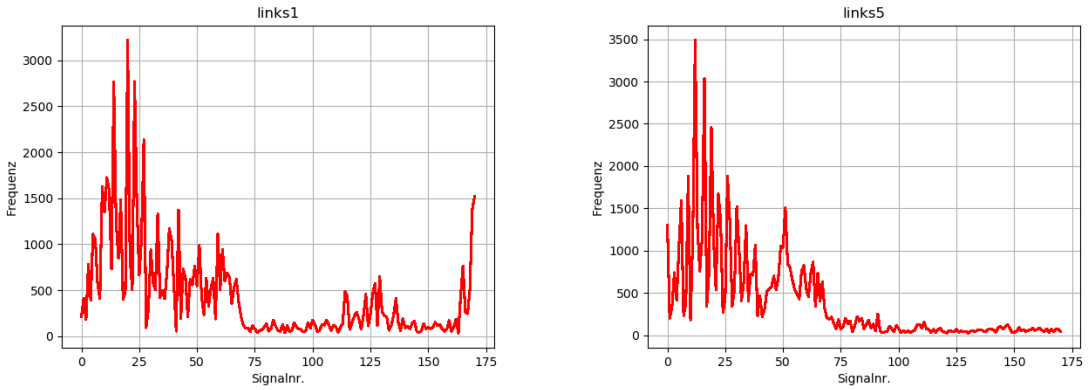


Abbildung 3.1: Sämtliche Signale mit Windowing berechnet

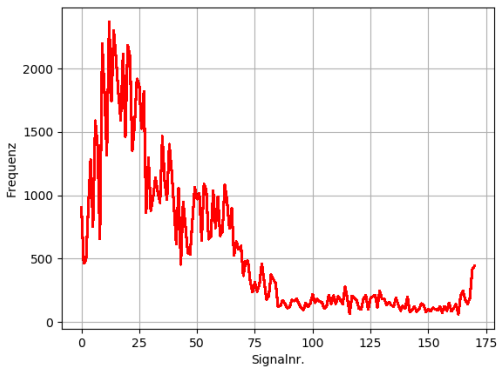


(a) Erste Sprachaufnahme von ’links’ (b) Fünfte Sprachaufnahme von ’links’

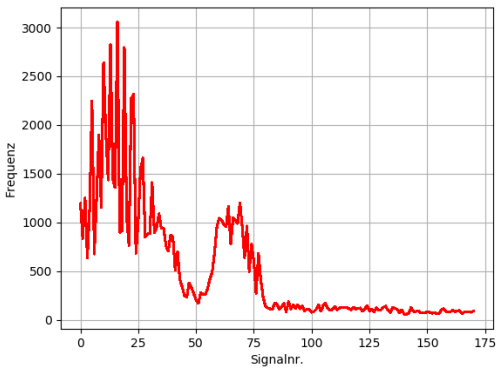
Abbildung 3.2: Zwei verschiedene Sprachaufnahmen im Vergleich.

3.3 Auswertung

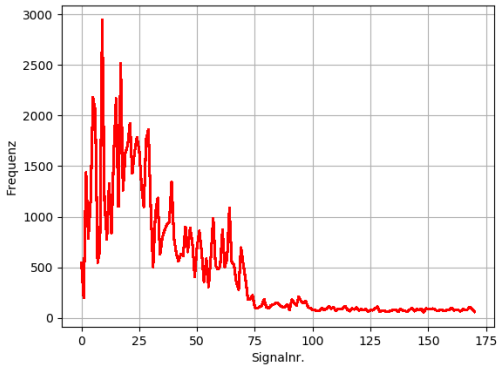
Die Daten aus den Numpydateien werden in einzelne Windows zerlegt mit einer Länge von 512 Samples die sich jeweils überlappen.
Die einzelnen Samples werden mit der Gaußschen Fensterfunktion multipliziert, welche eine Fensterbreite von 4 Standardabweichungen hat. In jedem Fenster wird eine lokale Fouriertransformation durchgeführt. Daraufhin werden die Fouriertransformierten über alle Fenster gemittelt.
Nachdem man sämtliche 4 Befehle und die jeweils fünf Beispiel dazu gemittelt hat, erhält man das Referenzspektrum.



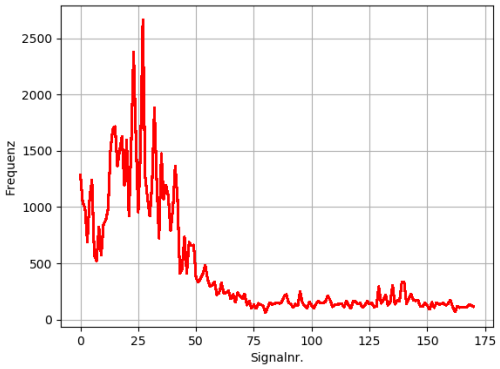
(a) Person 1: Referenzspektrum ’Links’



(b) Person 1: Referenzspektrum ’Rechts’



(c) Person 1: Referenzspektrum ’Hoch’



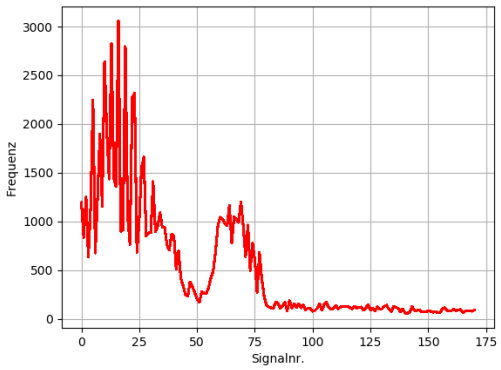
(d) Person 1: Referenzspektrum ’Tief’

Abbildung 3.3: Die vier Referenzspektren

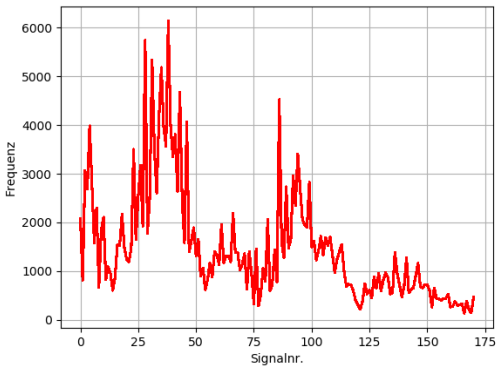
Hier werden zwei Eingabespektren beziehungsweise zwei Dateien mittels der Bravais-Pearson Methode verglichen auf ihre Korrelationskoeffizienten. Wenn die Spektren identisch sind, sollte der Wert 1.0 ergeben. Bei ungleichen Spektren sollte dieser nahe 0 sein. Person 1 ist die Person, welche die Spracheeingabe gemacht hat.

Spracheingabe	Person 1	Person 2
Hoch	0.5817	0.5496
Tief	0.7088	0.6172
Links	0.6514	0.6084
Rechts	0.4219	0.1779

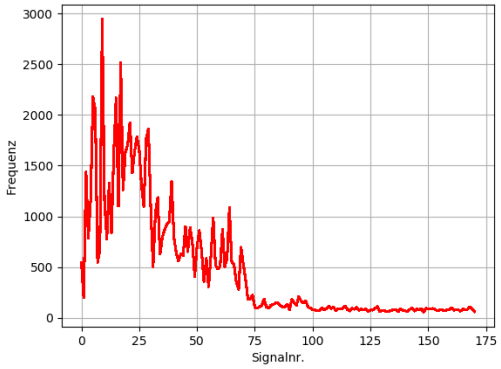
Tabelle 3.1: Vergleich der berechneten Referenzspektren



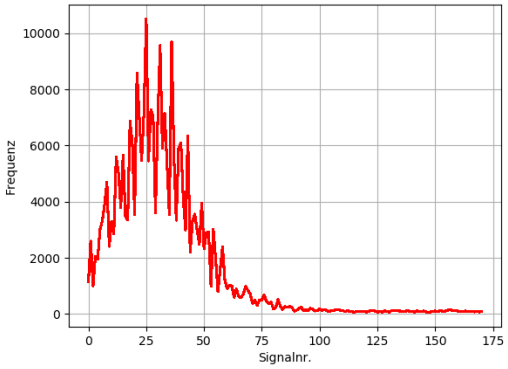
(a) Person 1: Referenzspektrum ’Rechts’



(b) Person 2: Referenzspektrum ’Rechts’



(c) Person 1: Referenzspektrum ’Hoch’



(d) Person 2: Referenzspektrum ’Hoch’

Abbildung 3.4: Die vier Referenzspektren mit den größten Unterschieden

3.4 Interpretation

Anhang

A.1 Quellcode

A.1.1 Quellcode Versuch 1

```
1  import pyaudio
2  import numpy
3  import matplotlib.pyplot as plt
4
5  FORMAT = pyaudio.paInt16
6  SAMPLEFREQ = 44100
7  FRAMESIZE = 1024
8  NOFFRAMES = 220
9  p = pyaudio.PyAudio()
10 print('running')
11
12 stream = p.open(format=FORMAT,
13               channels=1,
14               rate=SAMPLEFREQ,
15               input=True,
16               frames_per_buffer=FRAMESIZE)
17 data = stream.read(NOFFRAMES*FRAMESIZE)
18 decoded = numpy.fromstring(data, 'Int16');
19 numpy.save('aufgabe4/test.npy', decoded)
20
21 stream.stop_stream()
22 stream.close()
23 p.terminate()
```

Listing 4.1: Einlesen der Sprachaufnahme und ablegen des Signals in eine Numpy Datei

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

def time_axis(arr):
    return np.array(range(len(arr)))/44100

FORMAT = pyaudio.paInt16
SAMPLEFREQ = 44100
FRAMESIZE = 44100
NOFFRAMES = 2

p = pyaudio.PyAudio()
print('running')
stream = p.open(format=FORMAT, channels=1, rate=SAMPLEFREQ,
                input=True, frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES * FRAMESIZE)
decoded = np.fromstring(data, 'Int16') / ((2**15)/2-1)
stream.stop_stream()
stream.close()
p.terminate()

start = np.argmax(np.abs(decoded) > 0.05) - 1024
end = start + 44100
triggered = decoded[start:end]
triggered = np.concatenate((triggered, [0]*(44100 - end - start)))

np.savetxt("aufgabe4/tief_5_" + str(int(time.time())) + ".npy", triggered)
```

Listing 4.2: Einlesen einer Sprachaufnahme mit Aktivierung durch Triggerung

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

# Einlesen der .csv Datei
data = np.load('data/test.npy')
freq = np.zeros(225280)

# Darstellung des Amplitudenspektrums
plt.plot(data)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.savefig('data/img/testamp.png')
plt.show()

# Einlesen der .csv Datei
data2 = np.load('data/rechts2.npy')

# Darstellung des Amplitudenspektrums
plt.plot(data2)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.savefig('data/img/rechtsamp.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(data)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.xlim(50000, 75000)
plt.savefig('data/img/testamp2.png')
plt.show()

# Der zweite Wert wird absolut minus den ersten absoluten wert gerechnet um später den Wert
difference = 2 / 225280
# Die zweite Spalte der .csv Datei wird Fouriertransformiert
fourier = np.fft.fft(data[:225280])
# Die Fouriertransformierte Frequenz wird absolutiert, so dass kein negativer Wert mehr vorzufinden ist
spektrum = np.abs(fourier)
```

```
# Formel um die Anzahl der Schwingungen in die Freuquenz umzurechnen − f = n / (M ∗ t)
for x in range(0, 225280, 1):
    freq[x] = (x / (difference ∗ 225280))

# Darstellung des Amplitudenspektrums
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 60000)
plt.savefig('data/img/testspektrum1.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 35000)
plt.savefig('data/img/testspektrum2.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 1000)
plt.savefig('data/img/testspektrum3.png')
plt.show()
```

Listing 4.3: Amplitudenspektrum und Ausgabe von Plots

```
from scipy import signal

import matplotlib.pyplot as plt
import numpy as np

# Einlesen der .csv Datei
data = np.load('data/test.npy')
window = np.zeros((879, 512))
z = 256
freq = np.zeros(225280)
gaussianwindow = signal.windows.gaussian(512, std=4)
for y in range(0, 879):
    z = z - 256
    for x in range(0, 512):
        window[y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
        z = z + 1
    # plt.plot(window[y])
    # plt.title('Windownr' + str(y+1))
    # plt.xlabel('Signalnr.')
    # plt.ylabel('Frequenz')
    # plt.grid(True)
    # plt.savefig('data/img/' + str(y) + '.png')
    # plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(gaussianwindow)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/gauss.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(window)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/Alle.png')
plt.show()

for y in range(0, 879):
```

```
42      for x in range(0, 512):
43          window[y][x] = window[y][x] * gaussianwindow[x]
44      window[y] = np.abs(np.fft.fft(window[y]))
45      window[y] = np.mean(window[y])
46
47  plt.plot(window, 'r')
48  plt.grid(True)
49  plt.xlabel('Signalnr.')
50  plt.ylabel('Frequenz')
51  plt.savefig('data/img/AlleRichtig.png')
52  plt.show()
53
54  # Der zweite Wert wird absolut minus den ersten absoluten wert gerechnet um später den Wert
55  difference = 2 / 225280
56  # Die zweite Spalte der .csv Datei wird Fouriertransformiert
57  fourier = np.fft.fft(window)
58  # Die Fouriertransformierte Frequenz wird absolutiert, so dass kein negativer Wert mehr vorzufinden ist
59  spektrum = np.abs(fourier)
60  # Formel um die Anzahl der Schwingungen in die Freuquenz umzurechnen – f = n / (M * t)
61  for x in range(0, 225280, 1):
62      freq[x] = (x / (difference * 225280))
63
64  plt.plot(freq, 'r')
65  plt.grid(True)
66  plt.xlabel('Signalnr.')
67  plt.ylabel('Frequenz')
68  plt.savefig('data/img/ampwin.png')
69  plt.show()
```

Listing 4.4: Windowing und Ausgabe von Plots bzw. Windows

A.1.2 Quellcode Versuch 2

```
from scipy import signal

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Einlesen der .csv Datei

num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
    "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
num2 = ["ahoch1", "ahoch2", "ahoch3", "ahoch4", "ahoch5", "atief1", "atief2", "atief3", "atief4", "atief5",
    "alinks1", "alinks2", "alinks3", "alinks4", "alinks5", "arechts1", "arechts2", "arechts3", "arechts4",
    "arechts5"]
numm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
    "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
nummm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
    "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]
gaussianwindow = signal.windows.gaussian(512, std=4)

# Darstellung des Amplitudenspektrums
plt.plot(gaussianwindow)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/gauss.png')
plt.show()

for a in range(0, 20):
    data = np.load('data/' + str(num[a]) + '.npy')
    data2 = np.load('data/' + str(num2[a]) + '.npy')
    num[a] = np.zeros((171, 512))
    num2[a] = np.zeros((171, 512))
    z = 256

    for y in range(0, 171):
        z = z - 256
        for x in range(0, 512):
            num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
```

```
41            num2[a][y, x] = np.mean(np.abs(np.fft.fft(data2[z] * gaussianwindow)))
42            z = z + 1
43        # plt.plot(num[a][y])
44        # plt.title('Windownr' + str(y+1+(a*171)))
45        # plt.xlabel('Signalnr.')
46        # plt.ylabel('Frequenz')
47        # plt.grid(True)
48        # plt.savefig('data/img/' + str(y+1+(a*171)) + '.png')
49        # plt.show()
50
51    for y in range(0, 171):
52        for x in range(0, 512):
53            num[a][y, x] = num[a][y, x] * gaussianwindow[x]
54            num2[a][y, x] = num2[a][y, x] * gaussianwindow[x]
55        num[a][y] = np.abs(np.fft.fft(num[a][y]))
56        num2[a][y] = np.abs(np.fft.fft(num2[a][y]))
57        num[a][y] = np.mean(num[a][y])
58        num2[a][y] = np.mean(num2[a][y])
59
60    plt.plot(num[a], 'r')
61    plt.title(str(nummm[a]))
62    plt.grid(True)
63    plt.xlabel('Signalnr.')
64    plt.ylabel('Frequenz')
65    plt.savefig('data/img/' + numm[a] + 'AlleRichtig.png')
66    plt.show()
67
68 for z in range(0, 4):
69    capital[z] = np.zeros((171, 512))
70    capital2[z] = np.zeros((171, 512))
71    for y in range(0, 171):
72        for x in range(0, 512):
73            if (z == 0):
74                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
75                                    + num[4][y, x]) / 4
76                capital2[z][y, x] = (num2[0][y, x] + num2[1][y, x] + num2[2][y, x]
77                                     + num2[3][y, x] + num2[4][y, x]) / 4
78            elif (z == 1):
79                capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
80                                    + num[9][y, x]) / 4
81                capital2[z][y, x] = (num2[5][y, x] + num2[6][y, x] + num2[7][y, x] + num2[8][y, x]
82                                     + num2[9][y, x]) / 4
```

```
83          elif (z == 2):
84              capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
85                                  + num[14][y, x]) / 4
86              capital2[z][y, x] = (num2[10][y, x] + num2[11][y, x] + num2[12][y, x] + num2[13][y, x]
87                                   + num2[14][y, x]) / 4
88          elif (z == 3):
89              capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
90                                  + num[19][y, x]) / 4
91              capital2[z][y, x] = (num2[15][y, x] + num2[16][y, x] + num2[17][y, x] + num2[18][y, x]
92                                   + num2[19][y, x]) / 4
93
94      plt.plot(capital[z], 'r')
95      plt.grid(True)
96      plt.xlabel('Signalnr.')
97      plt.ylabel('Frequenz')
98      plt.savefig('data/img/' + capital2[z] + 'Average.png')
99      plt.show()
```

Listing 4.5: Windowing und Mittelung der Spektren

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Einlesen der .csv Datei

num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
      "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
numm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
       "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]

gaussianwindow = signal.windows.gaussian(512, std=4)

for a in range(0, 20):
    data = np.load('data/' + str(num[a]) + '.npy')
    num[a] = np.zeros((171, 512))
    z = 256

    for y in range(0, 171):
        z = z - 256
        for x in range(0, 512):
            num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
            z = z + 1

    for y in range(0, 171):
        for x in range(0, 512):
            num[a][y, x] = num[a][y, x] * gaussianwindow[x]
        num[a][y] = np.abs(np.fft.fft(num[a][y]))
        num[a][y] = np.mean(num[a][y])

for z in range(0, 4):
    capital[z] = np.zeros((171, 512))
    for y in range(0, 171):
        for x in range(0, 512):
            if (z == 0):
                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
                                    + num[4][y, x]) / 4
            elif (z == 1):
```

```
42              capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
43                      + num[9][y, x]) / 4
44          elif (z == 2):
45              capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
46                      + num[14][y, x]) / 4
47          elif (z == 3):
48              capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
49                      + num[19][y, x]) / 4
50      capital[z] = capital[z].ravel()
51
52  for x in range(0, 20):
53      num[x] = num[x].ravel()
54
55  r, p = scipy.stats.pearsonr(num[0], num[0])
56  print("r:", r, "p:", p)
57
58  r, p = scipy.stats.pearsonr(num[0], capital[0])
59  print("r:", r, "p:", p)
60
61  r, p = scipy.stats.pearsonr(num[0], capital[0])
62  print("r:", r, "p:", p)
```

Listing 4.6: Windowing und Bravais-Pearson Methode

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Einlesen der .csv Datei

num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
    "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
anderer = ["ahoch1", "ahoch2", "ahoch3", "ahoch4", "ahoch5", "atief1", "atief2", "atief3", "atief4", "atief5",
        "alinks1", "alinks2", "alinks3", "alinks4", "alinks5", "arechts1", "arechts2", "arechts3", "arechts4",
        "arechts5"]
moi = ["mhoch1", "mhoch2", "mhoch3", "mhoch4", "mhoch5", "mtief1", "mtief2", "mtief3", "mtief4", "mtief5",
    "mlinks1", "mlinks2", "mlinks3", "mlinks4", "mlinks5", "mrechts1", "mrechts2", "mrechts3", "mrechts4",
    "arechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]

gaussianwindow = signal.windows.gaussian(512, std=4)

for a in range(0, 20):
    data = np.load('data/' + str(num[a]) + '.npy')
    data2 = np.load('data/' + str(anderer[a]) + '.npy')
    data3 = np.load('data/' + str(moi[a]) + '.npy')
    num[a] = np.zeros((171, 512))
    moi[a] = np.zeros((171, 512))
    anderer[a] = np.zeros((171, 512))
    z = 256

    for y in range(0, 171):
        z = z - 256
        for x in range(0, 512):
            num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
            anderer[a][y, x] = np.mean(np.abs(np.fft.fft(data2[z] * gaussianwindow)))
            moi[a][y, x] = np.mean(np.abs(np.fft.fft(data3[z] * gaussianwindow)))
            z = z + 1

    for y in range(0, 171):
        for x in range(0, 512):
            num[a][y, x] = num[a][y, x] * gaussianwindow[x]
            anderer[a][y, x] = anderer[a][y, x] * gaussianwindow[x]
```

```
            moi[a][y, x] = moi[a][y, x] * gaussianwindow[x]
        num[a][y] = np.abs(np.fft.fft(num[a][y]))
        anderer[a][y] = np.abs(np.fft.fft(anderer[a][y]))
        moi[a][y] = np.abs(np.fft.fft(moi[a][y]))
        num[a][y] = np.mean(num[a][y])
        anderer[a][y] = np.mean(anderer[a][y])
        moi[a][y] = np.mean(moi[a][y])

for z in range(0, 4):
    capital[z] = np.zeros((171, 512))
    for y in range(0, 171):
        for x in range(0, 512):
            if (z == 0):
                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
                                    + num[4][y, x]) / 4
            elif (z == 1):
                capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
                                    + num[9][y, x]) / 4
            elif (z == 2):
                capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
                                    + num[14][y, x]) / 4
            elif (z == 3):
                capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
                                    + num[19][y, x]) / 4
    capital[z] = capital[z].ravel()

for x in range(0, 20):
    num[x] = num[x].ravel()
    moi[x] = moi[x].ravel()
    anderer[x] = anderer[x].ravel()


r1, p = scipy.stats.pearsonr(capital[0], moi[0])
print("capital−moi1 r:", r1, "p:", p)
r2, p = scipy.stats.pearsonr(capital[0], moi[1])
print("capital−moi2 r:", r2, "p:", p)
r3, p = scipy.stats.pearsonr(capital[0], moi[2])
print("capital−moi3 r:", r3, "p:", p)
r4, p = scipy.stats.pearsonr(capital[0], moi[3])
print("capital−moi4 r:", r4, "p:", p)
r5, p = scipy.stats.pearsonr(capital[0], moi[4])
print("capital−moi5 r:", r5, "p:", p)
```

```
84
85 s1, p = scipy.stats.pearsonr(capital[0], anderer[0])
86 print("capital−anderer1 r:", s1, "p:", p)
87 s2, p = scipy.stats.pearsonr(capital[0], anderer[1])
88 print("capital−anderer2 r:", s2, "p:", p)
89 s3, p = scipy.stats.pearsonr(capital[0], anderer[2])
90 print("capital−anderer3 r:", s3, "p:", p)
91 s4, p = scipy.stats.pearsonr(capital[0], anderer[3])
92 print("capital−anderer4 r:", s4, "p:", p)
93 s5, p = scipy.stats.pearsonr(capital[0], anderer[4])
94 print("capital−anderer5 r:", s5, "p:", p)
95
96 print()
97 r1 = (r1 + r2 + r3 + r4 + r5) / 5
98 print(r1)
99 s1 = (s1 + s2 + s3 + s4 + s5) / 5
100 print(s1)
101 print()
102 # −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
103
104 r6, p = scipy.stats.pearsonr(capital[1], moi[5])
105 print("capital−moi1 r:", r6, "p:", p)
106 r7, p = scipy.stats.pearsonr(capital[1], moi[6])
107 print("capital−moi2 r:", r7, "p:", p)
108 r8, p = scipy.stats.pearsonr(capital[1], moi[7])
109 print("capital−moi3 r:", r8, "p:", p)
110 r9, p = scipy.stats.pearsonr(capital[1], moi[8])
111 print("capital−moi4 r:", r9, "p:", p)
112 r10, p = scipy.stats.pearsonr(capital[1], moi[9])
113 print("capital−moi5 r:", r10, "p:", p)
114
115 s6, p = scipy.stats.pearsonr(capital[1], anderer[5])
116 print("capital−anderer1 r:", s6, "p:", p)
117 s7, p = scipy.stats.pearsonr(capital[1], anderer[6])
118 print("capital−anderer2 r:", s7, "p:", p)
119 s8, p = scipy.stats.pearsonr(capital[1], anderer[7])
120 print("capital−anderer3 r:", s8, "p:", p)
121 s9, p = scipy.stats.pearsonr(capital[1], anderer[8])
122 print("capital−anderer4 r:", s9, "p:", p)
123 s10, p = scipy.stats.pearsonr(capital[1], anderer[9])
124 print("capital−anderer5 r:", s10, "p:", p)
125
```

```
126 print()
127 r1 = (r6 + r7 + r8 + r9 + r10) / 5
128 print(r1)
129 s1 = (s6 + s7 + s8 + s9 + s10) / 5
130 print(s1)
131 print()
132 # ----------------------------------------------------------------------
133
134 r11, p = scipy.stats.pearsonr(capital[2], moi[10])
135 print("capital-moi1 r:", r11, "p:", p)
136 r12, p = scipy.stats.pearsonr(capital[2], moi[11])
137 print("capital-moi2 r:", r12, "p:", p)
138 r13, p = scipy.stats.pearsonr(capital[2], moi[12])
139 print("capital-moi3 r:", r13, "p:", p)
140 r14, p = scipy.stats.pearsonr(capital[2], moi[13])
141 print("capital-moi4 r:", r14, "p:", p)
142 r15, p = scipy.stats.pearsonr(capital[2], moi[14])
143 print("capital-moi5 r:", r15, "p:", p)
144
145 s11, p = scipy.stats.pearsonr(capital[2], anderer[10])
146 print("capital-anderer1 r:", s11, "p:", p)
147 s12, p = scipy.stats.pearsonr(capital[2], anderer[11])
148 print("capital-anderer2 r:", s12, "p:", p)
149 s13, p = scipy.stats.pearsonr(capital[2], anderer[12])
150 print("capital-anderer3 r:", s13, "p:", p)
151 s14, p = scipy.stats.pearsonr(capital[2], anderer[13])
152 print("capital-anderer4 r:", s14, "p:", p)
153 s15, p = scipy.stats.pearsonr(capital[2], anderer[14])
154 print("capital-anderer5 r:", s15, "p:", p)
155
156 print()
157 r1 = (r11 + r12 + r13 + r14 + r15) / 5
158 print(r1)
159 s1 = (s11 + s12 + s13 + s14 + s15) / 5
160 print(s1)
161 print()
162 # ----------------------------------------------------------------------
163
164 r16, p = scipy.stats.pearsonr(capital[3], moi[15])
165 print("capital-moi1 r:", r16, "p:", p)
166 r17, p = scipy.stats.pearsonr(capital[3], moi[16])
167 print("capital-moi2 r:", r17, "p:", p)
```

```
168 r18, p = scipy.stats.pearsonr(capital[3], moi[17])
169 print("capital−moi3 r:", r18, "p:", p)
170 r19, p = scipy.stats.pearsonr(capital[3], moi[18])
171 print("capital−moi4 r:", r19, "p:", p)
172 r20, p = scipy.stats.pearsonr(capital[3], moi[19])
173 print("capital−moi5 r:", r20, "p:", p)
174
175 s16, p = scipy.stats.pearsonr(capital[3], anderer[15])
176 print("capital−anderer1 r:", s16, "p:", p)
177 s17, p = scipy.stats.pearsonr(capital[3], anderer[16])
178 print("capital−anderer2 r:", s17, "p:", p)
179 s18, p = scipy.stats.pearsonr(capital[3], anderer[17])
180 print("capital−anderer3 r:", s18, "p:", p)
181 s19, p = scipy.stats.pearsonr(capital[3], anderer[18])
182 print("capital−anderer4 r:", s19, "p:", p)
183 s20, p = scipy.stats.pearsonr(capital[3], anderer[19])
184 print("capital−anderer5 r:", s20, "p:", p)
185
186 print()
187 r1 = (r16 + r17 + r18 + r19 + r20) / 5
188 print(r1)
189 s1 = (s16 + s17 + s18 + s19 + s20) / 5
190 print(s1)
191 print()
```

Listing 4.7: Bravais-Pearson Methode mit Ausgabe der Korrelation