



**Hochschule Konstanz**  
Technik, Wirtschaft und Gestaltung

**Signale, Systeme und Sensoren**

# **Fourieranalyse und Akustik**

**T. Schoch, L. Stratmann**

**Konstanz, 12. Mai 2019**

## Zusammenfassung (Abstract)

Thema:	Fourieranalyse und Akustik	
Autoren:	T. Schoch	tobias.schoch@htwg-konstanz.de
	L. Stratmann	luca.stratmann@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz Jürgen Keppler Christoph Kaiser	mfranz@htwg-konstanz.de juergen.keppler@htwg-konstanz.de christoph.kaiser@htwg-konstanz.de

In dem dritten Versuch der Versuchsreihe werden wir die Fourieranalyse auf akustische Signale und Systeme in Form von Musik durch eine Mundharmonika und Rückkopplung anwenden. Die Signale werden auf einem Oszilloskop angezeigt.

In den beiden Teilen des Versuchs werden wir akustische Signale mittels der Python Bibliothek TekTDS2000 aufnehmen und abspeichern. Im Anschluss erfolgt mittels Python die Auswertung der Messdaten.

Dabei werden die Techniken der Fouriertransformation und des Bode Diagramms.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>IV</b>
<b>Listingverzeichnis</b>	<b>V</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Versuch 1</b>	<b>2</b>
2.1 Fragestellung, Messprinzip, Aufbau, Messmittel . . . . .	2
2.2 Messwerte . . . . .	5
2.3 Auswertung . . . . .	6
2.4 Interpretation . . . . .	9
<b>3 Versuch 2</b>	<b>10</b>
3.1 Fragestellung, Messprinzip, Aufbau, Messmittel . . . . .	10
3.2 Messwerte . . . . .	10
3.3 Auswertung . . . . .	10
3.4 Interpretation . . . . .	10
<b>Anhang</b>	<b>11</b>
A.1 Quellcode . . . . .	11
A.1.1 Quellcode Versuch 1.1 . . . . .	11
A.1.2 Quellcode Versuch 1.2 . . . . .	12
A.1.3 Quellcode Versuch 1.3 . . . . .	12
A.1.4 Quellcode Versuch 2.1 . . . . .	14
A.1.5 Quellcode Versuch 2.2 . . . . .	16
A.2 Messergebnisse . . . . .	20

# **Abbildungsverzeichnis**

2.1	Versuchsaufbau Teil 1 . . . . .	3
2.2	Signal einer Mundharmonika . . . . .	5
2.3	Das Amplitudenspektrum von der Fourieranalyse . . . . .	7
3.1	Einlesen des Signals in .csv und durch das Oszilloskop . . . . .	10
3.1a	Die Ausgabe des Oszilloskopes . . . . .	10
3.1b	Die Auswertung der .csv Datei durch Python . . . . .	10
4.2	Messergebnisse für Task 2 . . . . .	20

# **Tabellenverzeichnis**

2.1	Zu berechnende Werte	.....	6
2.2	Zu berechnende Werte	.....	8

# **Listingverzeichnis**

4.1	Das Bild des Oszilloskopes einlesen und abspeichern . . . . .	11
4.2	.csv Datei einlesen und in einem Plot graphisch wiedergeben . . . . .	12
4.3	Fouriertransformation anwenden und ein Amplitudenspektrum ausgeben sowie Berechnung einiger Werte . . . . .	12
4.4	Lautsprecherdaten graphisch ausgeben und einige Werte berechnen . . . . .	14
4.5	Amplitude Phasenverschiebung und Bode-Diagramm berechnen . . . . .	16

# 1

## Einleitung

[? ] [? ]

# 2

## Versuch 1

### 2.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im ersten Versuch der Versuchsreihe "Fourieranalyse und Akustik" werden wir die Fourieranalyse auf akustische Signale und Systeme anwenden. Dazu werden wir mittels einer Mundharmonika in ein Mikrofon blasen, dass wiederum an ein Oszilloskop angeschlossen ist und die Spannungen anzeigt. Dabei ist es wichtig zu beachten, dass auf dem Oszilloskop mehrere Perioden abgebildet werden. Das Triggerlevel soll so im Oszilloskop eingestellt werden, dass das Signal nur bei genügend hoher Amplitude eingestellt wird. Die Triggerung sollte dabei auch im Single Sequence Modus eingestellt werden um statische Daten zu erhalten die auch dem tatsächlichen Spannungswert entsprechen. Nachdem dies erledigt ist, haben wir ein Python Skript geschrieben task1.1.py um das Signal aus dem Oszilloskop bei der Einstellung der Triggerung SSingle Sequenzäuszulesen. Dies wird mittels der Toolbox TekTDS2000 von M. Miller in eine .csv Datei gespeichert welche zum Beispiel mit Excel geöffnet werden kann.

So sieht der Versuchsaufbau des ersten Versuches aus. Dabei wird das Mikrofon an den Channel 1 des Oszilloskopes angeschlossen. Nach der Kalibrierung und der Einstellung der künstlichen Sinuskurve haben wir mit dem Python Skript das ausgegebene Signal des Oszilloskopes in eine .csv Datei eingelesen. Danach hatten wir die Aufgabe folgende Werte zu berechnen.

- Grundperiode (in ms)
- Grundfrequenz (in Hz)
- Signaldauer (in s)
- Abtastfrequenz (in Hz)
- Signallänge M (Anzahl der Abtastzeitpunkte)
- Abtastintervall  $\Delta t$  (in s)



Abbildung 2.1: Versuchsaufbau Teil 1

Im Anschluss mussten wir mithilfe der Funktion `numpy.fft.fft()` die Fouriertransformation des Signals berechnen. Daraus sollten wir das Amplitudenspektrum bestimmen und grafisch darstellen. Dabei ist zu beachten, dass die x-Achse mit der Frequenz nicht in Hertz sondern in der Anzahl Schwingungen innerhalb der gesamten Signaldauer definiert ist.. Die Frequenz  $f$  in Hertz lässt sich jedoch folgendermaße berechnen.

$$f = \frac{n}{M * \Delta t}$$

Als letzten Teil der ersten Aufgabe sollen wir die Grundfrequenz im Spektrum berechnen und damit die Frequenz in Hertz identifizieren. Folgende Materialien wurden benötigt:

- Oszilloskop
- Mikrofon
- Mundharmonika
- Signalkabel
- Python auf einem Computer

## 2.2 Messwerte

Auf dem Bild ist das Signal abgebildet, dass wir mittels einer Mundharmonika in das Mikrofon geblasen haben.

Durch die Toolbox TekTDS2000 können wir das Signal nun in eine .csv Datei umwandeln und abspeichern.

Diese Datei haben wir mittels Numpy und der Funktion `np.loadtxt` ausgelesen. Die Ergebnisse wurden in die 2 verschiedene Funktionen `x` und `y` geschrieben.

Danach wurden diese mittels matplotlib graphisch dargestellt.

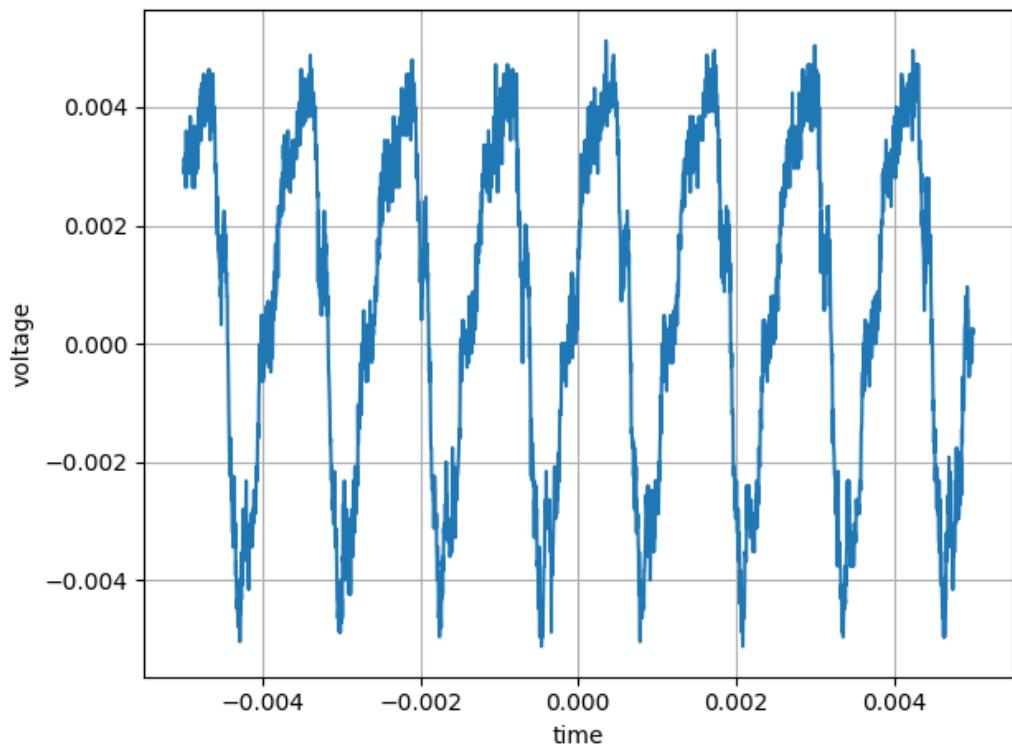


Abbildung 2.2: Signal einer Mundharmonika

## 2.3 Auswertung

Auf dem Spannungsverlauf sieht man das aufgenommene Signal der Mundharmonika. Durch einen Pythonfunktion die wir aus der Toolbox TekTDS2000 erhalten haben, konnten wir den Spannungsverlauf aus dem Oszilloskop einlesen und in eine .csv Datei schreiben.

Dieses Format wird auch häufig für Excel Dateien verwendet. Durch das Drücken des Knopfes Single Sequence haben wir auf dem Oszilloskop ein Standbild erhalten.

Diese Datei haben wir mittels Numpy und der Funktion `np.loadtxt` ausgelesen. Die Ergebnisse wurden in die 2 verschiedene Funktionen `x` und `y` geschrieben.

Danach wurden diese mittels matplotlib graphisch dargestellt.

Die Grundperiode und die Grundfrequenz haben wir uns mittels der Toolbox TekTDS2000 durch `getFreq(1)` und `getPeriod(1)` ausgeben lassen. Die Signallänge M haben wir durch `len(file.readlines())` erfahren.

Durch die absolute Addition von den beiden Maxima- und Minimawerten erhalten wir gerundet die Signaldauer. Wenn man nun die Signaldauer dividiert durch die Signallänge erhält man das Abtastintervall  $\Delta t$  mit dem Wert  $4\mu s$ .

Mit dem Abtastintervall erhält man auch die Abtastfrequenz. Diese entspricht 250000 Hertz.

Im folgenden sind die Ergebnisse der Berechnungen aufzufinden:

Plot	Wert
Grundperiode (in ms)	1.275ms
Grundfrequenz (in Hertz)	784.31Hz
Signaldauer	0,01s
Abtastfrequenz (in Kilohertz)	250KHz
Signallänge M (Anzahl der Abtastzeitpunkte)	2500
Abtastintervall $\Delta t$ (in $\mu s$ )	$4\mu s$

Tabelle 2.1: Zu berechnende Werte

Das Amplitudenspektrum erhält man indem man die Spannung absolut fouriertransformiert mit der Numpy Funktion `np.fft.fft()` und als Amplitude auf der y Achse ausgibt.

Doch bisher wird die x-Achse in der Einheit *Anzahl Schwingungen innerhalb der gesamten Signaldauer* und nicht in Hertz angegeben.

Um dies jedoch zu tun müssen wir die Frequenz in f folgendermaßen berechnen:

$$f = \frac{n}{M * \Delta t}$$

Die jeweilige Schwingung n wird geteilt durch die Signallänge und das Abtastintervall. Dadurch erhält man folgenden Graphen, welcher bis zu dem von uns festgelegten Frequenzwert von 20000 geplottet wird.

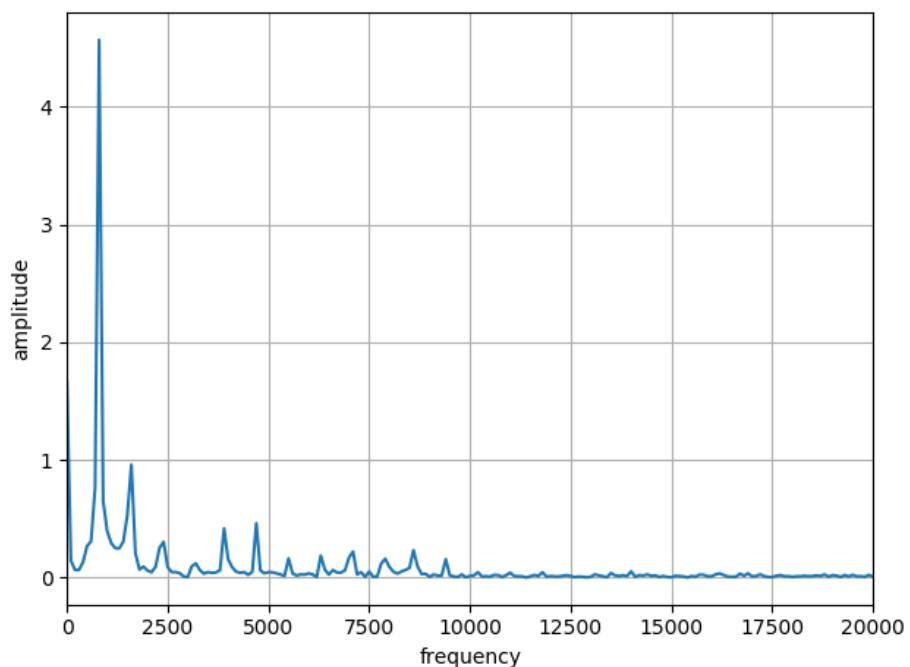


Abbildung 2.3: Das Amplitudenspektrum von der Fourieranalyse

Mittels der Numpy Funktion `np.max(spektrum)` können wir den maximalen Amplitudenwert abfragen. Mittels einer for-Schleife fragen wir nun den Wert bis zum 1250sten Signal ab der das Maxima hat.

Dadurch erhalten wir die Frequenz des maximalen Amplitudenwerts. Wir fragen genau bis zur Hälfte der Werte ab, da der Graph sich spiegelt und es so 2 Maxima gibt. Da wir nur den ersten Wert wollen müssen wir dies festlegen.

Im folgenden sind die Ergebnisse der Berechnungen aufzufinden:

Plot	Wert
Maximaler Amplitudenwert	4.5675
Frequenz des Maxima	800Hz

Tabelle 2.2: Zu berechnende Werte

## 2.4 Interpretation

Da das Signal der Mundharmonika schön gleichmäßig ist, kann man feststellen, dass der Ton sehr gleichmäßig gespielt wurde.

Durch die Beschaffenheit des Funktionsweise und die laute Gegebenheiten zum Zeitpunkt der Messung, kann man gut beobachten, dass die Amplitude nach so gut wie jeder periodischen Wiederholung zunimmt und damit die Spannung auch höher wird. Im großen Ganzen jedoch gibt es keine Unstimmigkeiten.

Für die Tabelle 2.1 kann man folgende Schlussfolgerungen ziehen:

- Die Signaldauer ist komplementär mit der Signallänge und dem Abtastintervall
- Das Abtastintervall ist komplementär zur Abtastfrequenz
- Frequenz und Zeit sind komplementär zueinander

Bei der Abbildung 2.3 ist eine schöne Amplitude zu erkennen, die sich von den anderen Amplituden durch ihre Höhe absetzt. Dieser maximale Amplitudenausschlag mit der Höhe 4.5675V ist der Ton.

Der Ton den wir produziert haben liegt also demnach bei 800Hz. Da wir bis auf eine Öffnung die Anderen zugeklebt haben, wissen wir dass wir einen Ton mit der Frequenz von 800Hz gespielt haben.

Die anderen kleinen Harmonischen sind die Obertöne. Wir wissen durch das Amplitudenspektrum auch, dass wir keinen Klang gespielt haben und die Mundharmonika richtig zugeklebt haben.

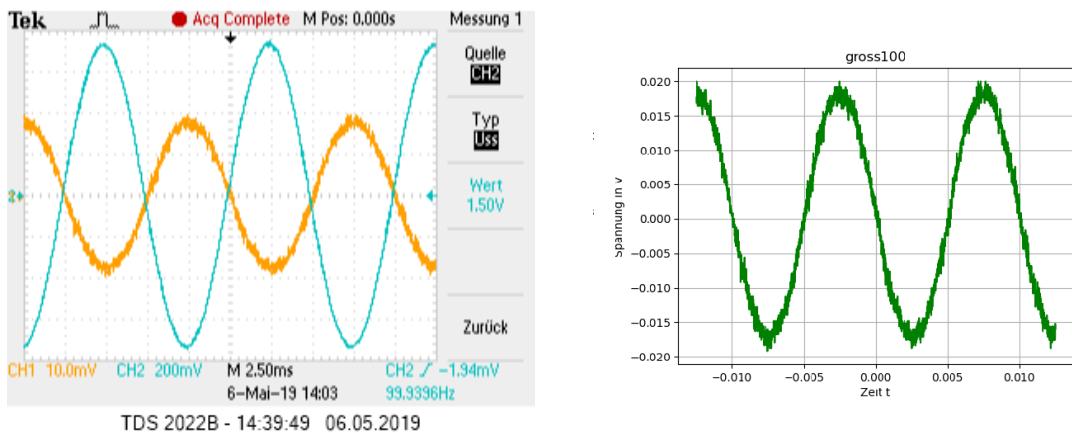
Wenn man die Fouriertransformierte für alle 2500 Abtastungen graphisch darstellt, dann stellt sich eine y-Achsenspiegelung dar. Da wir jedoch keine Achsenspiegelung wollen sondern ledigliche bis Frequenzen von 20000Hz kürzen wir die x-Achse um nur die wichtigen Werte zu erlangen.

# 3

## Versuch 2

### 3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

### 3.2 Messwerte



(a) Die Ausgabe des Oszilloskopes

(b) Die Auswertung der .csv Datei durch Python

Abbildung 3.1: Einlesen des Signals in .csv und durch das Oszilloskop

### 3.3 Auswertung

### 3.4 Interpretation

# Anhang

## A.1 Quellcode

### A.1.1 Quellcode Versuch 1.1

```
1 from TekTDS2000 import *
2
3 scope = TekTDS2000()
4
5 # Einlesen vom Channel 1 und vom Channel 2
6 scope.saveCsv(filename='versuch3/kleinerLautsprecher/100.csv', ch=1)
7 scope.saveCsv(filename='100_2.csv', ch=2)
8
9 frequency = scope.getFreq(1)
10 period = scope.getPeriod(1)
11
12 print("Frequenz", frequency)
13 print("Periode", period)
```

Listing 4.1: Das Bild des Oszilloskopes einlesen und abspeichern

## A.1.2 Quellcode Versuch 1.2

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Einlesen der .csv Datei
5 x, y = np.loadtxt('data/eins.csv', delimiter=',', unpack=True)
6
7 # Darstellung des Signals unserer Mundharmonikaaufnahme
8 plt.plot(x * 1000, y * 1000, 'b')
9 plt.ylabel('Spannung in mV')
10 plt.xlabel('Zeit in ms')
11 plt.grid(True)
12 plt.show()
```

Listing 4.2: .csv Datei einlesen und in einem Plot graphisch wiedergeben

## A.1.3 Quellcode Versuch 1.3

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Einlesen der .csv Datei
5 data = np.genfromtxt('data/eins.csv', delimiter=',', skip_header=0, skip_footer=0)
6 # 2 Zeilen aus der linken Spalte in time schreiben
7 time = data[:, 0]
8 # Der zweite Wert wird absolut minus den ersten absoluten wert gerechnet um später den Wert
9 difference = np.abs(time[1] - time[0])
10 # Die zweite Spalte der .csv Datei wird Fouriertransformiert
11 fourier = np.fft.fft(data[:, 1])
12 # Die Fouriertransformierte Frequenz wird absolutiert, so dass kein negativer Wert mehr vorzufinden ist
13 spektrum = np.abs(fourier)
14 # Formel um die Anzahl der Schwingungen in die Freqeuenz umzurechnen – f = n / (M * t)
15 freq = range(0, 2500, 1) / (difference * 2500)
16
17 # Darstellung des Amplitudenspektrums
18 plt.plot(freq, spektrum)
19 plt.grid()
20 plt.xlabel('frequency')
21 plt.ylabel('amplitude')
22 plt.xlim(0, 20000)
23 plt.show()
```

```

24
25 # Einlesen der Signallänge
26 file = open("data/eins.csv")
27 signallaenge = len(file.readlines())
28 # Sekundenumwandlung so wird 0,000001s zu 1s
29 sek = 1000000
30 # Das Abtastintervall in s anzeigen und runden
31 abtastintervall = round((difference * sek), 2)
32
33 # For-Schleife um den passenden Frequenzwert zu erlangen der zu der maximalen Amplitude gehört
34 for x in range(0, 1250):
35     if round(spektrum[x], 4) == 4.5675:
36         frequency = freq[x]
37
38 # Berechnung der größten Amplitude
39 print("Grundperiode: 0.001275 s", )
40 print("Grundfrequenz: 784.31 Hz", )
41 print()
42 print("Signaldauer: ", (abtastintervall * signallaenge) / sek, " s")
43 print("Abtastfrequenz: ", 1 / abtastintervall * sek, " Hz")
44 print("Signallänge M: ", signallaenge)
45 print("Abtastintervall t:", abtastintervall, " s")
46 print()
47 print("Maximalster Amplitudenausschlag", round(frequency, 1))
48 print("Frequenz mit der größten Amplitude", np.max(spektrum))

```

Listing 4.3: Fouriertransformation anwenden und ein Amplitudenspektrum ausgeben sowie Berechnung einiger Werte

## A.1.4 Quellcode Versuch 2.1

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Array um die einzelnen .csv Dateien einzulesen des kleinen Lautsprechers
5 klein = ["klein100", "klein200", "klein300", "klein400", "klein500", "klein700", "klein850", "klein1000",
6         "klein1200", "klein1500", "klein1700", "klein2000", "klein3000", "klein4000", "klein5000", "klein6000",
7         "klein10000"]
8 # Array um die einzelnen .csv Dateien einzulesen des großen Lautsprechers
9 gross = ["gross100", "gross200", "gross300", "gross400", "gross500", "gross700", "gross850", "gross1000",
10        "gross1200", "gross1500", "gross1700", "gross2000", "gross3000", "gross4000", "gross5000", "gross6000",
11        "gross10000"]
12
13 #for-Schleife für die 17 unterschiedlichen .csv Dateien
14 for a in range(0, 17):
15     # Einlesen der kleinen Dateien – dabei wird die linke Spalte als x und die rechte als y definiert
16     x, y = np.loadtxt('data/' + klein[a] + '.csv', delimiter=',', unpack=True)
17     # Einlesen der großen Dateien – dabei wird die linke Spalte als u und die rechte als v definiert
18     u, v = np.loadtxt('data/' + gross[a] + '.csv', delimiter=',', unpack=True)
19
20     # Die einzelnen .csv Dateien des kleinen Lautsprechers werden in jeweils einem Plot dargestellt
21     plt.plot(x, y, 'b')
22     plt.title(klein[a])
23     plt.ylabel('Spannung in V')
24     plt.xlabel('Zeit t')
25     plt.grid(True)
26     plt.savefig(str(klein[a]) + '.png')
27     plt.show()
28
29     # Die einzelnen .csv Dateien des großen Lautsprechers werden in jeweils einem Plot dargestellt
30     plt.plot(u, v, 'g')
31     plt.title(gross[a])
32     plt.ylabel('Spannung in V')
33     plt.xlabel('Zeit t')
34     plt.grid(True)
35     plt.savefig(str(gross[a]) + '.png')
36     plt.show()
37
38     # Die ersten beiden Zeiten für den kleinen Lautsprecher werden in time1 geschrieben
39     time1 = x[:2, ]
40     # Die ersten beiden Zeiten für den großen Lautsprecher werden in time1 geschrieben
```

```

41 time2 = u[:2, ]
42
43 # Die zwei Zeiten des kleinen Lautsprechers werden subtrahiert und multipliziert für eine mikrosekunden Darstellung
44 # Zudem wird die berechnete Dauer gerundet
45 timing1 = round((time1[1] - time1[0]) * 100000, 5)
46 # Die zwei Zeiten des großen Lautsprechers werden subtrahiert und multipliziert für eine mikrosekunden Darstellung
47 # Zudem wird die berechnete Dauer gerundet
48 timing2 = round((time2[1] - time2[0]) * 100000, 5)
49
50 # Ausgeben einer Tabelle im LaTeX Format
51 print("\hline")
52 # Der maximale Amplitudenwert des kleinen Lautsprechers für die jeweilige Frequenz
53 print("Amplitude:", klein[a], np.max(np.abs(y)))
54 # Der maximale Amplitudenwert des großen Lautsprechers für die jeweilige Frequenz
55 print("Amplitude:", gross[a], np.max(np.abs(v)))
56 print("ms", klein[a], timing1)
57 print("ms", gross[a], timing2)
58
59 print("\hline")

```

Listing 4.4: Lautsprecherdaten graphisch ausgeben und einige Werte berechnen

## A.1.5 Quellcode Versuch 2.2

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Array um die einzelnen .csv Dateien einzulesen des kleinen Lautsprechers
5 klein = ["klein100", "klein200", "klein300", "klein400", "klein500", "klein700", "klein850", "klein1000",
6         "klein1200", "klein1500", "klein1700", "klein2000", "klein3000", "klein4000", "klein5000", "klein6000",
7         "klein10000"]
8 # Array um die einzelnen .csv Dateien einzulesen des großen Lautsprechers
9 gross = ["gross100", "gross200", "gross300", "gross400", "gross500", "gross700", "gross850", "gross1000",
10        "gross1200", "gross1500", "gross1700", "gross2000", "gross3000", "gross4000", "gross5000", "gross6000",
11        "gross10000"]
12
13 # Die für jeweils den großen und den kleinen Lautsprecher gemessenen Zeiten
14 zeit = [100, 200, 300, 400, 500, 700, 850, 1000, 1200, 1500, 1700, 2000, 3000, 4000, 5000, 6000, 10000]
15 # Die für den kleinen Lautsprecher von Hand berechneten Phasenverschiebungen in s
16 kleinphase = [228, 295, 113.2, 105.2, 70.6, 64.8, 34, 4.2, 1.8, 12.2, 12.1, 14.7, 8.88, 11.60, 5.92, 6.92, 3.67]
17 # Die für den großen Lautsprecher von Hand berechneten Phasenverschiebungen in s
18 grossphase = [505, 82, 1, 10, 17, 12.4, 15.2, 14.8, 15.6, 12.8, 13.2, 14.4, 19.7, 12.52, 5.32, 3.32, 4.74]
19 # Die einzelnen Abtastintervalle für die einzelnen Frequenzen des großen Lautsprechers
20 grossintervall = [1, 1, 0.4, 0.4, 0.4, 0.4, 0.4, 0.2, 0.2, 0.1, 0.1, 0.1, 0.04, 0.04, 0.04, 0.02, 0.01]
21 # Die einzelnen Abtastintervalle für die einzelnen Frequenzen des kleinen Lautsprechers
22 kleinintervall = [1, 1, 1, 1, 1, 0.4, 0.4, 0.4, 0.4, 0.2, 0.2, 0.2, 0.1, 0.04, 0.04, 0.04, 0.02]
23 # Ein Vektor in dem später für den kleinen Lautsprecher die Amplituden gespeichert werden
24 kleinamp = np.zeros(17)
25 # Ein Vektor in dem später für den großen Lautsprecher die Amplituden gespeichert werden
26 grossamp = np.zeros(17)
27
28 # For Schleife von 0–16 um die einzelnen .csv Dateien einzulesen und auszuwerten
29 for a in range(0, 17):
30     # Einlesen der kleinen Dateien – dabei wird die linke Spalte als x und die rechte als y definiert
31     x, y = np.loadtxt('data/' + klein[a] + '.csv', delimiter=',', unpack=True)
32     # Einlesen der großen Dateien – dabei wird die linke Spalte als u und die rechte als v definiert
33     u, v = np.loadtxt('data/' + gross[a] + '.csv', delimiter=',', unpack=True)
34
35     # Berechnung für jede Datei des kleinen Lautsprechers den maximalen absoluten Amplitudenwert
36     kleinamp[a] = np.max(np.abs(y))
37     # Berechnung für jede Datei des großen Lautsprechers den maximalen absoluten Amplitudenwert
38     grossamp[a] = np.max(np.abs(v))
39
40 # Darstellung der Amplitudenmaxima für beide Lautsprecher im Verhältnis zur Dateinr.
```

```

41 plt.plot(kleinamp, 'b')
42 plt.plot(grossamp, 'y')
43 plt.title("Amplitude")
44 plt.ylabel('Amplitude')
45 plt.xlabel('Versuchnr.')
46 plt.grid(True)
47 plt.savefig('amplitudeanzahl.png')
48 plt.show()
49
50 # Darstellung der Amplitudenmaxima für beide Lautsprecher im Verhältnis zur Frequenz
51 plt.plot(zeit, kleinamp, 'b')
52 plt.plot(zeit, grossamp, 'y')
53 plt.title("Amplitude")
54 plt.ylabel('Amplitude')
55 plt.xlabel('Frequenz f')
56 plt.grid(True)
57 plt.savefig('amplitudefrequenz.png')
58 plt.show()
59
60 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Dateinr.
61 plt.plot(kleinphase, 'b')
62 plt.plot(grossphase, 'y')
63 plt.title("Phasengang")
64 plt.ylabel('Phasenverschiebung')
65 plt.xlabel('Versuchnr.')
66 plt.grid(True)
67 plt.savefig('phasenanzahl.png')
68 plt.show()
69
70 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Frequenz
71 plt.plot(zeit, kleinphase, 'b')
72 plt.plot(zeit, grossphase, 'y')
73 plt.title("Phasenverschiebung")
74 plt.ylabel('Phasenverschiebung')
75 plt.xlabel('Frequenz f')
76 plt.grid(True)
77 plt.savefig('phasenfrequenz.png')
78 plt.show()
79
80 #for Schleife zum Berechnen des Bode Diagramms
81 for a in range(0, 17):
82     # 20 log10 zur Berechnung des Bode-Diagramms für den großen Lautsprecher

```

```

83 grossamp[a] = 1 / grossamp[a]
84 grossamp[a] = 20 * np.log10(grossamp[a])
85 # 20 log10 zur Berechnung des Bode-Diagramms für den großen Lautsprecher
86 kleinamp[a] = 1 / kleinamp[a]
87 kleinamp[a] = 20 * np.log10(kleinamp[a])
88 # Berechnung des Phasenwinkels mit t*f*360 für den großen Lautsprecher
89 grossphase[a] = (grossphase[a] * -1) * zeit[a] * 360
90 # Berechnung des Phasenwinkels mit t*f*360 für den kleinen Lautsprecher
91 kleinphase[a] = (kleinphase[a] * -1) * zeit[a] * 360
92
93 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Dateinr.
94 plt.plot(kleinamp, 'b')
95 plt.plot(grossamp, 'y')
96 plt.title("Bode-Amplitude")
97 plt.ylabel('Amplitude')
98 plt.xlabel('Versuchnr.')
99 plt.grid(True)
100 plt.semilogx()
101 plt.savefig('bodeamplitudeanzahl.png')
102 plt.show()
103
104 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Frequenz
105 plt.plot(zeit, kleinamp, 'b')
106 plt.plot(zeit, grossamp, 'y')
107 plt.title("Bode-Amplitude")
108 plt.ylabel('Amplitude')
109 plt.xlabel('Frequenz f')
110 plt.grid(True)
111 plt.semilogx()
112 plt.savefig('bodeamplitudefrequenz.png')
113 plt.show()
114
115 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Dateinr.
116 plt.plot(kleinphase, 'b')
117 plt.plot(grossphase, 'y')
118 plt.title("Bode-Phasengang")
119 plt.ylabel('Phasenverschiebung')
120 plt.xlabel('Versuchnr.')
121 plt.grid(True)
122 plt.semilogx()
123 plt.savefig('bodephasenanzahl.png')
124 plt.show()

```

```

125
126 # Darstellung des Phasengangs für beide Lautsprecher im Verhältnis zur Frequenz
127 plt.plot(zeit, kleinphase, 'b')
128 plt.plot(zeit, grossphase, 'y')
129 plt.title("Bode–Phasenverschiebung")
130 plt.ylabel('Phasenverschiebung')
131 plt.xlabel('Frequenz f')
132 plt.grid(True)
133 plt.semilogx()
134 plt.savefig('bodephasenfrequenz.png')
135 plt.show()

```

Listing 4.5: Amplitude Phasenverschiebung und Bode-Diagramm berechnen

## A.2 Messergebnisse

	klein ms	Abstand intervall	Phasen- verschiebung	ms	Abstand- intervall	Phasen- verschiebung
100	25ms	1 ms	228 μs	25ms	1 ms	505 μs
200	25ms	1 ms	185 μs	25ms	1 ms	82 μs
300	10 ms	0,4 μs	113,2 μs	25ms	1 μs	1 μs
400	10 ms	0,6 μs	105,2 μs	25ms	1 μs	10 μs
500	10 ms	0,4 μs	70,6 μs	25ms	1 μs	17 μs
700	10 ms	0,4 μs	64,8 μs	10 ms	0,4 μs	12,4 μs
950	10ms	0,4μs	34 μs	10 ms	0,4 μs	15,2 μs
1000	5 ms	0,2 μs	4,2 μs	10 ms	0,4 μs	14,8 μs
1200	5ms	0,2 μs	1,8 μs	10ms	0,4 μs	15,6 μs
1500	2,5ms	0,1 μs	1,2,2 μs	5 ms	0,2 μs	12,3 μs
1700	2,5ms	0,1 μs	12,1 μs	5 ms	0,2 μs	13,2 μs
2000	2,5ms	0,1 μs	14,7 μs	5ms	0,2 μs	14,4 μs
3000	1 ms	0,04 μs	8,88 μs	2,5ms	0,1 μs	19,7 μs
4000	1 ms	0,04 μs	11,60 μs	1ms	0,04 μs	12,52 μs
5000	1ms	0,04μs	5,92 μs	1ms	0,04 μs	5,32 μs
6000	500 μs	0,02 μs	6,92 μs	1ms	0,04 μs	3,32 μs
10000	250 μs	0,01 μs	3,67 μs	0,5ms	0,02 μs	4,74 μs

Kleine  
Lautsprecher
Große  
Lautsprecher

Abbildung 4.2: Messergebnisse für Task 2