GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

PHYSIKALISCHES
GRUNDPRAKTIKUM

VERSUCH 1

# DER POHLSCHE RESONATOR

*Praktikant:*
Tobias Wegener
Alexander Osterkorn

*E-Mail:*
tobias.wegener@stud.uni-goettingen.de
a.osterkorn@stud.uni-goettingen.de

*Tutor:*
Marten Düvel

*Gruppe:*
3

*Durchgeführt am:*
24.6.2013

*Protokoll abgegeben:*
1.7.2013

Testiert:

# Inhaltsverzeichnis

# 1 Einleitung

In diesem Versuch, benannt nach dem berühmten Göttinger Professor für Experimentalphysik, *Robert Wichard Pohl*, werden die Gesetzmäßigkeiten von freier und getriebener Schwingung und Resonanz untersucht.

# 2 Theorie

## 2.1 Freie Schwingung

Die Bewegung des Pohlschen Rades *ohne äußeren Antrieb* lässt sich durch die Gleichung einer allgemeinen harmonischen Schwingung mit Dämpfung beschreiben. Dafür betrachtet man die Erhaltung des Drehmoments und erhält

$$I\ddot{\varphi} + \rho\dot{\varphi} + D\varphi = 0 \tag{1}$$

Dabei ist $I$ das Trägheitsmoment des Rades, $\rho$ der Reibungskoeffizient für lineare Reibung und $D$ die Winkelrichtgröße, die für ein rücktreibendes Drehmoment sorgt, das proportional zur momentanen Auslenkung $\varphi$ ist.
Zur weiteren Berechnung normiert man die Gleichung durch Division mit $I$ und führt neue Variablen ein:

$$\ddot{\varphi} + 2\beta\dot{\varphi} + \omega_0^2\varphi = 0 \tag{2}$$

Das wird allgemein gelöst durch [Demtröder, 2013, S.333]:

$$\varphi(t) = \varphi_0 \mathrm{e}^{-\beta t}\mathrm{e}^{i(\omega_e t-\phi)} \quad \text{bzw.} \quad \varphi(t) = \varphi_0 \mathrm{e}^{-\beta t}\cos(\omega_e t - \phi) \tag{3}$$

$\omega_e$ heißt *Eigenfrequenz* und berechnet sich [ebd.] zu

$$\omega_e^2 = \omega_0^2 - \beta^2 \tag{4}$$

Das *logarithmische Dekrement* $\Lambda$ [Demtröder, 2013, S.334] wird definiert als

$$\Lambda = \log\left(\frac{\varphi(t)}{\varphi(t+T)}\right) = \log\left(\mathrm{e}^{\beta T}\frac{\mathrm{e}^{i(\omega_e t-\phi)}}{\mathrm{e}^{i(\omega_e(t+T)-\phi)}}\right) = \beta T \tag{5}$$

## 2.2 Erzwungene Schwingung

Etwas komplizierter wird die Situation, wenn die Schwingung trotz Reibung durch eine äußere Drehmomentzuführung aufrechterhalten wird. Die Gleichung 2 erhält dann eine *Inhomogenität* mit irgendeiner Konstanten $N$:

$$\ddot{\varphi} + 2\beta\dot{\varphi} + \omega_0^2\varphi = N\cos(\omega t) \tag{6}$$

Es ist eine spezielle Lösung dieser DGL gesucht. Physikalisch kann man sich überlegen, dass die Schwingung nach einer gewissen Einschwingphase mit der Anregungsfrequenz $\omega$ schwingt. Also ist der entsprechende Lösungsanstz $\varphi(t) = \frac{\varphi_0}{2}\mathrm{e}^{i(\omega t-\phi)}$.
Man erhält [Demtröder, 2013, S.336] für die Amplitude $\varphi_0$ dieser Schwingung

$$\varphi_0 = \frac{N}{\sqrt{(\omega_0^2-\omega^2)^2 + 4\beta^2\omega^2}} \tag{7}$$

und für die Phasenverschiebung $\phi$ zwischen Schwingung und Anregung gilt:

$$\phi = \arctan\left(\frac{2\beta\omega}{\omega_0^2-\omega^2}\right) \tag{8}$$

Die Diskussion von Amplitude und Phasenverschiebung sollen hier nicht durchgeführt werden.
Es ist aber zum einen noch wichtig zu erwähnen, dass die maximale Amplitude in Gl. 7 bei $\omega = \omega_r = \sqrt{\omega_0^2 - 2\beta^2}$ maximal wird, was man durch Ableiten bestätigen kann.
Bei der Phasenverschiebung ist zu erwähnen, dass sie im Allgemeinen immer negativ ist, was hier heißen soll, dass die Schwingung der Anregung hinterherläuft [Demtröder, 2013, S.336]. Hier wurde ein Lösungsanstz für die inhomogene Gleichung mit $-\phi$ gewählt. Im Bereich $\omega \leq \omega_0$ gilt damit $0 \leq \phi \leq \frac{\pi}{2}$ und für $\omega_0 \leq \omega$ gilt $\frac{\pi}{2} \leq \phi \leq \pi$. Eine grafische Ausarbeitung dieser Tatsache für verschiedene Dämpfungen folgt dann im Auswertungsteil.

# 3 Durchführung

## 3.1 Versuchsaufbau

Der Kern des Versuchsaufbaus zum Pohlschen Resonators ist eine Schwungscheibe, welche über eine Spiralfeder eine Schwingungsbewegung ausführen kann. An einer Winkelskala lässt sich die Amplitude der Schwingung ablesen. Vervollständigt wird der Aufbau durch eine Wirbelstrombremse, welche eine Dämpfung der Schwingung bewirkt und einem Schrittmotor, welcher die Schwingung extern antreiben kann. Dieser Motor ist über eine Steuerungsanlage mit einem Computer verbunden, sodass die Messung vom Computer gesteuert werden kann. Auch die Datenaufnahme erfolgt komplett elektronisch. Die Stärke der Wirbelstrombremse lässt sich am Versuchsaufbau variieren.

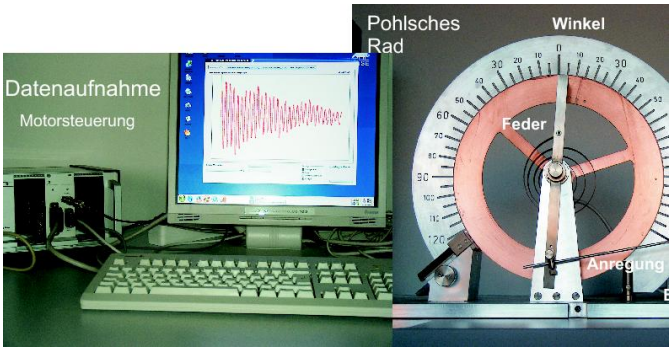


Abbildung 1: Versuchsanordnung

## 3.2 Aufnahme der Messdaten

### 3.2.1 Freie, gedämpfte Schwingung

Im ersten Teil der Messung wird für verschiedene Dämpfungswerte eine Schwingungskurve aufgenommen. Die Stärke der Dämpfung lässt sich über ein kleines Rad an der Wirbelstrombremse variieren. Dort sollen nacheinander die Werte $0, 4, 6 und 8$mm eingestellt werden. Für jede dieser Einstellungen wird dann das Schwungrad um 120° ausgelenkt und die abklingende Schwingung aufgenommen. Da diese Schwingung frei sein soll, wird am Computer die Anregungsfrequenz auf $\omega = 0$Hz eingestellt.

### 3.2.2 Erzwungene, gedämpfte Schwingung

Auch im zweiten Teil der Messung wiederholt man eine Messroutine für verschiedene Dämpfungswerte (Stellung am Milimetertrieb der Wirbelstrombremse: $4, 6 und 8$mm). Für jeden dieser Werte variiert man nun die Anregungsfrequenz so, dass das Intervall $100 - 600$Hz mit einer geeigneten Schrittweite durchlaufen wird. Eine solche Schrittweite könnte z.B. 50Hz sein. Im interessanten Bereich in der Nähe der Eigenfrequenz sollte diese jedoch verkleinert werden, um aussagekräftigere Ergebnisse zu bekommen. Die Eigenfrequenz ist dann erreicht, wenn die Amplitude maximal ist. Für jede dieser Frequenzen wartet man dann zunächst den Einschwingvorgang ab, sodass sich eine stationäre Schwingung einstellt. Sobald dies der Fall ist, startet man eine neue Messung und nimmt einige Schwingungsperioden mit dem Programm auf.

# 4 Auswertung

## 4.1 Freie, gedämpfte Schwingung

### 4.1.1 Abklingkurven

In Messung 1 wurde eine freie gedämpfte Schwingung für verschiedene Dämpfungswerte über mehrere Perioden betrachtet. Da die Schwingung erst etwas zeitversetzt zu dem Programmstart gestartet wurde, werden die bis dahin registrierten Werte entfernt. Außerdem sind die gemessenen Zeiten nicht relativ zum Beginn der Messung gegeben, sodass die x-Werte um denjenigen x-Wert, bei dem die Messung begonnen hat, verschoben werden.

Diese Daten können nun geplottet werden. Da in diesem Auswertungsteil besonders die Dämpfung betrachtet werden soll, ist es sinnvoll, zusätlich die Einhüllende zu plotten. Dazu werden jeweils die Maxima (Minima) bestimmt und exponentiell mit gnuplot gefittet. Die grafische Darstellung dieser Messungen ist in Abbildung 2 zu sehen.

### 4.1.2 Charakteristische Schwingungs- und Dämpfungsgrößen

Für jeden Dämpfungswert sollen die Eigenfrequenz $\omega_e$, das logarithmische Dekrement $\Lambda$ sowie die Dämpfungskonstante $\beta$ bestimmt werden.

Die Eigenfrequenz $\omega_e$ lässt sich aus der Periodendauer $T$ berechnen. Es wird also zunächst einmal ein Wert für die Periodendauer $T$ benötigt. Dazu wird die Differenz der x-Werte zweier aufeinanderfolgender Maxima (bzw. Minima) für alle Extrema bestimmt. Aus diesen Werten lässt sich dann eine mittlere Periodendauer $T$ berechnen. Die Dämpfung sollte keinen Einfluss auf die Periodendauer haben, sodass diese während der Schwingungsmessung als konstant angenommen werden kann. Damit lässt sich nun die Eigenfrequenz berechnen:

$$\omega_e = \frac{2\pi}{T}$$
$$\sigma_{\omega_e} = \frac{2\pi}{T^2} \cdot \sigma_T$$

Für das logarithmische Dekrement $\Lambda$ gilt die Formel

$$\Lambda = \ln\left(\frac{\phi(t)}{\phi(t+T)}\right)$$

.

Auch dieses wird zunächst für alle Extrema ausgerechnet und dann gemittelt. Die Dämpfungskonstante $\beta$ ergibt sich dann direkt aus $\Lambda$:

$$\beta = \frac{\Lambda}{T}$$
$$\sigma_\beta = \sqrt{\left(\frac{1}{T} \cdot \sigma_\Lambda\right)^2 + \left(\frac{\Lambda}{T^2} \cdot \sigma_T\right)^2}$$

Die Ergebnisse dieser Berechnungen für jeden Dämpfungswert sind in Tabelle 1 aufgeführt.

| Dämpfung in mm | 0 | 4 | 6 | 8 |
|---|---|---|---|---|
| $T$ in s | $3,07 \pm 0,06$ | $3,1 \pm 0,1$ | $3,1 \pm 0,06$ | $3,23 \pm 0,15$ |
| $\omega_e$ in Hz | $2,04 \pm 0,04$ | $2,06 \pm 0,07$ | $2,03 \pm 0,04$ | $1,94 \pm 0,09$ |
| $\Lambda$ | $0,05 \pm 0,01$ | $0,5 \pm 0,1$ | $0,9 \pm 0,1$ | $1,4 \pm 0,4$ |
| $\beta$ in Hz | $0,017 \pm 0,003$ | $0,15 \pm 0,04$ | $0,29 \pm 0,03$ | $0,43 \pm 0,11$ |
| $\omega_0$ in Hz | $2,04 \pm 0,06$ | $2,1 \pm 0,1$ | $2,05 \pm 0,06$ | $2,00 \pm 0,12$ |

Tabelle 1: Übersicht über gemessene Schwingungsgrößen

#### 4.1.3 Ungedämpfte Eigenfrequenz

Die Eigenfrequenz $\omega_e$ hängt von der jeweiligen Dämpfungskonstante $\beta$ ab. Es kann jedoch aus diesen Werte die ungedämpfte Eigenfrequenz $\omega_0$ berechnet werden. Diese sollte im Idealfall recht gut mit der gemessenen Eigenfrequenz in der Stellung "0"der Wirbelstrombremse übereinstimmen. Diese Frequenz lässt sich wie folgt berechnen:

$$\omega_0 = \sqrt{\omega_e^2 + \beta^2}$$

$$\sigma_{\omega_0} = \sqrt{\left(\frac{1}{\omega_0} \cdot \omega_e \cdot \sigma_{\omega_e}\right)^2 + \left(\frac{1}{\omega_0} \cdot \beta \cdot \sigma_\beta\right)^2}$$

Auch diese Werte sind für jeden Dämpfungswert in Tabelle 3.1 aufgeführt. Da diese Größe später noch in der Auswertung benötigt wird, ist es sinnvoll, hier den gewichteten Mittelwert zu bilden:

$$\omega_0 = (2,05 \pm 0,04)\,\text{Hz}$$

Dieser Wert stimmt sehr gut mit dem Wert für die Eigenfrequenz bei der Messung ohne zusätzliche Dämpfung überein.

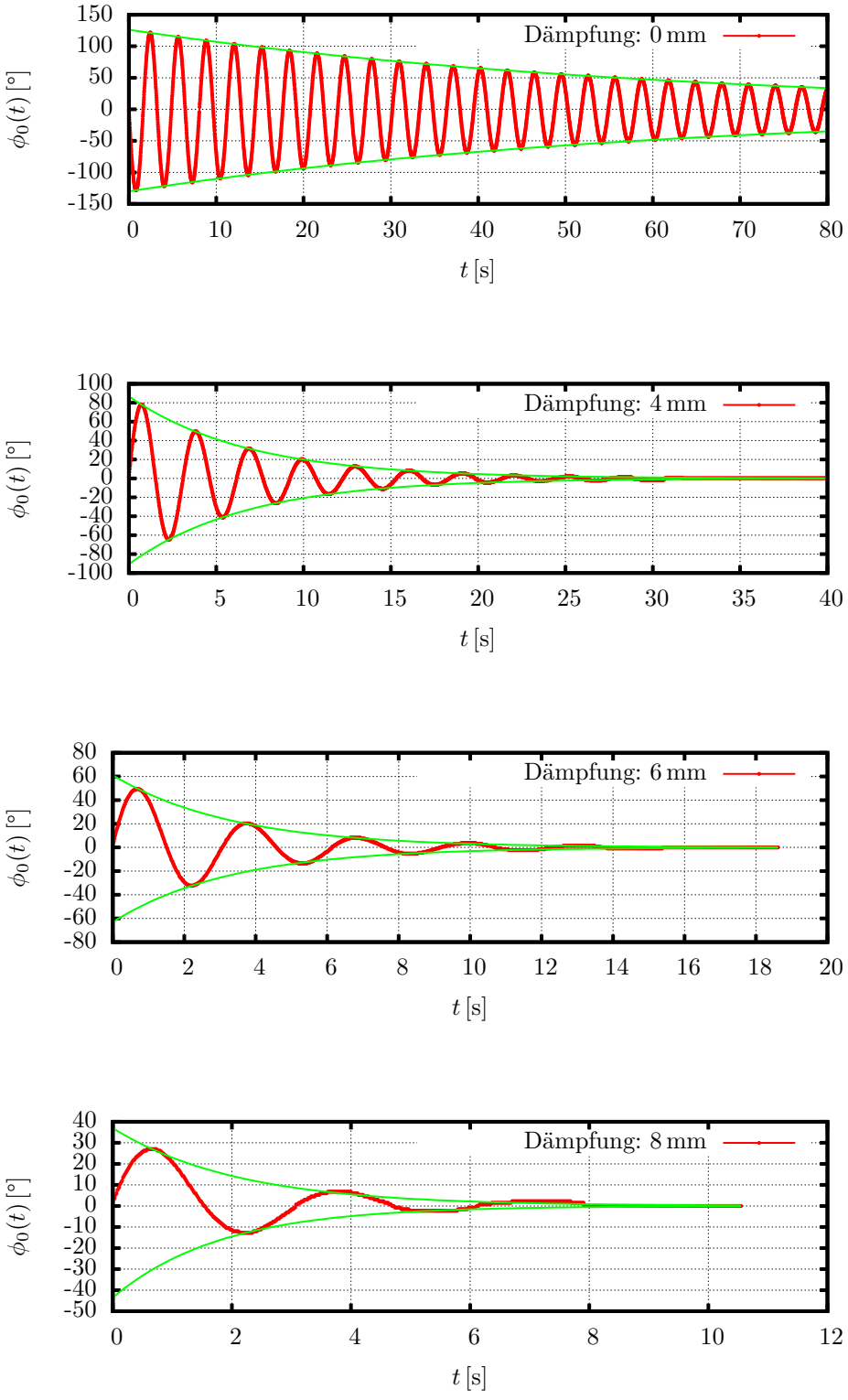


Abbildung 2: Abklingkurven aus Messung 1

## 4.2 Erzwungene, gedämpfte Schwingung

### 4.2.1 Resonanzkurven

Für jeden Dämpfungswert lässt sich nun eine Resonanzkurve bestimmen. Dazu wird die auf der Ordinate die Amplitude in Abhängigkeit von der Anregungsfrequenz $\frac{\phi(\omega(f))}{\phi(0)}$ (normiert durch den Nullausschlag) und auf der Abzisse die Anregungsfrequenz (mit Eigenfrequenz normiert) $\frac{\omega}{\omega_0}$ aufgetragen. Für $\phi_0$ wird $\phi(\omega(f))$ bei der kleinsten Anregungsfrequenz verwendet, da die Amplitude sich für kleine Frequenzen ("weit"von der Resonanzfrequenz entfernt) kaum ändert.

Es muss also für jeden Datensatz die Schwingungsamplitude bestimmt werden. Dazu müssen auch diese Daten erst einmal normiert werden, da sie im Allgemeinen etwas in y-Richtung verschoben sind, sodass sie keine symmetrische Schwingung um die Nulllage repräsentieren. Dazu bildet man für jeden Datensatz das arithm. Mittel der Maxima sowie das der Minima und verschiebt dann den Datensatz um das Mittel aus diesen beiden Werten. Nun kann der Betrag der y-Werte an den Extremstellen als Amplitude für die jeweilige Anregungsfrequenz verwendet werden.

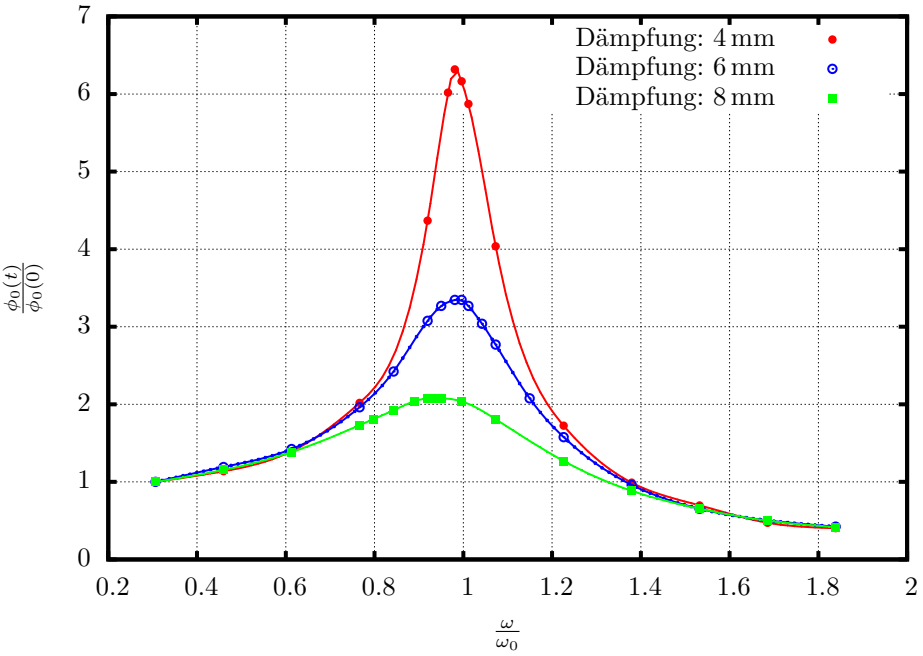


Abbildung 3: Frequenzgang einer erzwungenen gedämpften Schwingung für verschiedene Dämpfungen

### 4.2.2 Phasenverschiebung

Aus den Messwerten lässt sich außerdem für jeden Dämpfungswert der Gang der Phasenverschiebung in Abhängigkeit von der Anregungsfrequenz darstellen. Dazu muss zunächst einmal für jeden Datensatz (bestimmte Anregungsfrequenz und Dämpfung) die Phasenverschiebung bestimmt werden. In den Rohdaten sind diejenigen Zeitwerte, bei denen die Anregung einen -/+ Nulldurchgang hat, mit einer '1' gekennzeichnet. Da die Schwingung des Schwungrads der Anregungsschwingung hinterherläuft, muss also der nächste Zeitpunkt bestimmt werden, bei dem das Schwungrad einen -/+ Nulldurchgang aufweist. Die dazu verwendete Rechenroutine befindet sich im Anhang. Auch für die Phasenverschiebungen kann ein Dia-

gramm erstellt werden, indem die normierte Phasenverschiebung $frac\Phi(\omega)\pi$ gegen die normierte Anregungsfrequenz (s.o.) aufgetragen wird. Da die Phasenverschiebung zunächst in ° angegeben ist, muss sie noch in Bogenmaß umgerechnet werden.

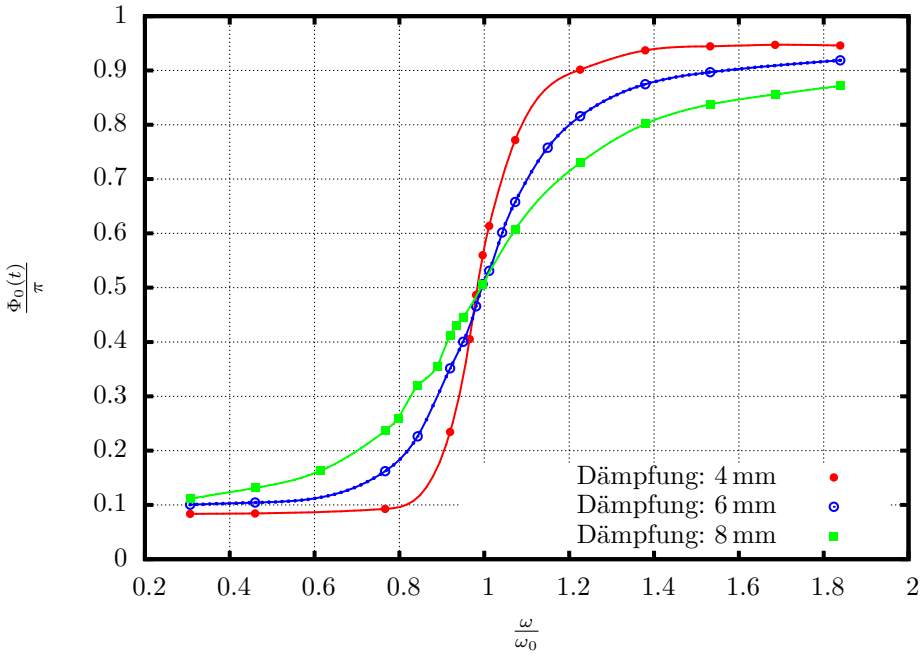


Abbildung 4: Phasenverschiebung einer getriebenen Schwingung für verschiedene Dämpfungen

### 4.2.3 Resonanzfrequenz

Aus den in Versuch 1 bestimmten Werten für die Eigenfrequenz $\omega_0$ und der Dämpfungskonstante $\beta$ lässt sich die Resonanzfrequenz $\omega_R$ theoretisch berechnen:

$$\omega_E = \sqrt{\omega_0^2 - 2\beta^2}$$

| Dämpfung in mm | 4 | 6 | 8 |
|---|---|---|---|
| $\omega_R$ in Hz (berechnet) | $2,04 \pm 0,04$ | $2,00 \pm 0,04$ | $1,95 \pm 0,06$ |
| $\omega_R$ in Hz (abgelesen) | $2,03 \pm 0,05$ | $2,03 \pm 0,05$ | $1,94 \pm 0,09$ |

Tabelle 2: Berechnete und abgelesene Resonanzfrequenzen für verschiedene Dämpfungen

Die berechneten Werte stimmen sehr gut mit den abgelesenen Werten überein.

# 5 Diskussion

Die im Experiment gefundenen Ergebnissen entsprechen den theoretischen Überlegungen in vollem Maße. Das Zustandekommen dieser guten Werte ist sicherlich auch darauf zurückzuführen, dass der Versuch weitesgehend automatisiert ist, sodass menschliche Fehler vermieden werden.

# Literatur

Wolfgang Demtröder. *Experimentalphysik 1.* Springer Spektrum, 2013.

# Anhang

```
/*-----------------------------------Analyse.cpp-----------------------------------
Programm zur Auswertung Pohlscher Resonator: Erstellt zu einer Datei, die
einen Schwingungsdatensatz enthaelt, drei neue Dateien mit den Maxima (*.max),
Minima (*.min) und Nulldurchgaengen (*.zero)
---------------------------------------------------------------------------------*/

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>

using namespace std;


//Globals:
const char* filename;
const char* outputFile;
vector<float> xValues;
vector<float> yValues;
vector<float> xMaxima;
vector<float> yMaxima;
vector<float> xMinima;
vector<float> yMinima;
vector<float> xZeroCross;
vector<float> yZeroCross;
vector<float> xExitCross;
char expInfo[5];

//Functions:
void readFile();          // Daten aus Datei einlesen
void findMaxima();        // Maxima finden und zwischenspeichern
void printMaxima();       // Maxima in Datei schreiben
void findMinima();        // Minima finden und zwischenspeichern
void printMinima();       // Minima in Datei schreiben
void findZeroCross();     // Nulldurchgaenge finden und zwischenspeichern
void printZeroCross();    // Nulldurchgaenge in Datei schreiben
float calcPhase();        // Phasenverschiebung berechnen
void printPhase();        // Phasenverschiebung in Datei schreiben
void getExpInfo();        // Versuchsdaten bekommen (Daempfung, Frequenz)



//[...]
```

```
/*-------------------------------void readFile()-------------------------------
Einlesen der Daten aus Datei und Zwischenspeicherung in jeweiligem vector
-----------------------------------------------------------------------------*/


void readFile()
{
        ifstream fin;
        fin.open(filename);

        if (fin.fail())
        {
                cerr << "Unable to open File." << endl;
        }

        // Lese die ersten zwei Zeilen ein (-> Text) ohne Verarbeitung

        string temp;
        getline (fin, temp, '\n');

        if (!(temp == "normierte Daten"))
        {
                getline(fin, temp, '\n');
        }

        // Zwischenspeicher fuer Werte in aktueller Zeile

        float current[3];

        while (!fin.eof())
        {
                fin >> current[0];
                fin >> current[1];
                fin >> current[2];

                // Werte aus Zeile in vector ablegen

                if (current[2] == 1)
                {
                        xExitCross.push_back(current[0]);
                }

                xValues.push_back(current[0]);
                yValues.push_back(current[1]);
        }

        fin.close();
}
```

```
/*-------------------------------void findMaxima()-------------------------------
Betrachte fuer jeden y-Wert die 40 vorherigen und die 40 folgenden. Falls
keiner dieser Werte kleiner ist, gibt es ein Maximum. Ausnahmen: Zu Beginn sind
die y-Werte konstant 0, erfuellen also die Bedingung und muessen ausgeschlossen
werden. Ausserdem: In der Naehe eines Maximums kann der Maximalwert mehrmals
hintereinander angenommen werden, hier soll nur der mittlere Wert registriert
werden. ----------------------------------------------------------------------*/

void findMaxima()
{
        bool check = true;

        for (int i=40; i<yValues.size()-40; i++)
        {
                // Anfangs-Offset und Mehrfachmaxima ueberspringen
                if (yValues[i] <= 0.01 || yValues[i] == yValues[i-1])
                {
                        continue;
                }

                // Vergleich mit umgebenden y-Werten
                for (int j=-40; j<40; j++)
                {
                        check = true; // reset

                        // kein max, falls anderer Wert aus Umgebung groesser
                        if (yValues[i] < yValues[i+j])
                        {
                                check = false;
                                break;
                        }
                }

                // kein Umgebungswert groesser ->  max gefunden
                if (check)
                {
                        /* falls y-Wert mehrfach hintereinander angenommen wird,
                        speicher nur ein Wertepaar aus der Mitte */

                        int counter = 1;

                        while (yValues[i] == yValues[i + counter])
                        {
                                counter++;
                        }

                        xMaxima.push_back(xValues[i + counter/2]);
                        yMaxima.push_back(yValues[i + counter/2]);
                }
        }

}
```

```
/*-------------------------------void findMinima()-------------------------------
Struktur analog zu findMaxima(), einige Ungleichungen werden umgedreht
-------------------------------------------------------------------------------*/


void findMinima()
{
        bool check = true;

        for (int i=40; i<yValues.size()-40; i++)
        {
                //Anfangs-Offset und Mehrfachminima ueberspringen
                if (yValues[i] >= 0.01 || yValues[i] == yValues[i-1])
                {
                        continue;
                }

                // Vergleich mit umgebenden y-Werten
                for (int j=-40; j<40; j++)
                {
                        check = true; // reset

                        // kein min, falls anderer Wert aus Umgebung kleiner
                        if (yValues[i] > yValues[i+j])
                        {
                                check = false;
                                break;
                        }
                }

                // kein Umgebungswert kleiner -> min gefunden
                if(check)
                {
                        /* falls y-Wert mehrfach hinereinander angenommen wird,
                        speicher nur ein Wertepaar aus der Mitte */
                        int counter = 1;
                        while (yValues[i] == yValues[i + counter])
                        {
                                counter++;
                        }

                        xMinima.push_back(xValues[i + counter/2]);
                        yMinima.push_back(yValues[i + counter/2]);
                }
        }
}
```

```
/*-------------------------------void normalize()-------------------------------
Falls offset vorhanden, werden die y-Werte konstant verschoben, sodass die Daten
eine symmetrische Schwingung um Null darstellen:
- Berechne arithmet. Mittel der Maxima und Minima
- Das arithmet. Mittel aus den beiden Ergebnissen sollte im Idealfall Null sein,
  andernfalls erhaelt man den offset-Wert
- Verschiebung der y-Werte um offset-Wert und Schreiben in neue Datei (*.norm)
------------------------------------------------------------------------------*/

void normalize()
{
        // Berechne offset

        string outputfile = (string) filename + ".norm";

        float offset;
        float maxCount = 0;
        float minCount = 0;

        for (int i=0; i<yMaxima.size(); i++)
        {
                maxCount += yMaxima[i];
        }

        for (int i=0; i<yMinima.size(); i++)
        {
                minCount += yMinima[i];
        }

        maxCount = maxCount / (float) yMaxima.size();
        minCount = minCount / (float) yMinima.size();

        offset = (maxCount + minCount) / 2;

        // Schreibe normierte Daten in neue Datei (*.norm)

        ofstream fout;
        fout.open (outputfile.c_str());

        fout << "normierte Daten" << endl;

        for (int i=0; i<xValues.size(); i++)
        {
                yValues[i] -= offset;
                fout << xValues[i] << "\t" << yValues[i]
                        << "\t" << exitValues[i] << endl;
        }

        fout.close();
}
```

```
/*-------------------------------void printMaxima()-------------------------------
Schreibt die Maxima in eine neue Datei (.max)
--------------------------------------------------------------------------------*/

void printMaxima()
{
        string outputFile = (string) (filename) + ".max";

        ofstream fout;
        fout.open(outputFile.c_str());

        for (int i=0; i < xMaxima.size(); i++)
        {
                fout << xMaxima[i] << "\t" << yMaxima[i] << endl;
        }

        fout.close();
}
```

```
/*-------------------------------void printMinima()-------------------------------
Schreibt die Minima in eine neue Datei (.min)
--------------------------------------------------------------------------------*/

void printMinima()
{
        string outputFile = (string) (filename) + ".min";

        ofstream fout;
        fout.open(outputFile.c_str());

        for (int i=0; i<xMinima.size(); i++)
        {
                fout << xMinima[i] << "\t" << yMinima[i] << endl;
        }

        fout.close();
}
```

```
/*-------------------------------void findZeroCross()-------------------------------
Finde Nulldurchgaenge (einen pro Periode, suche nach Vorzeichenwechsel -/+) und
speichere sie z
------------------------------------------------------------------------------*/

void findZeroCross()
{
        bool check = true;

        for (int i=20; i<yValues.size()-1; i++)
        {
                if (yValues[i] == yValues[i+1])
                {
                        continue;
                }

                if ((yValues[i] == 0 || (yValues[i] * yValues[i+1] < 0)) &&
                                (yValues[i-20] < 0 ))
                {
                        xZeroCross.push_back(xValues[i]);
                        yZeroCross.push_back(yValues[i]);
                }
        }
}




/*-------------------------------void printZeroCross()-------------------------------
Schreibe die Nullstellen in neue Datei (.zero)
------------------------------------------------------------------------------*/

void printZeroCross()
{
        string outputFile = string(filename) + ".zero";

        ofstream fout;
        fout.open(outputFile.c_str());

        for (int i=0; i<xZeroCross.size(); i++)
        {
                fout << xZeroCross[i] << "\t" << xExitCross[i] << "\t" << yZeroCross[i] <<
        }

        fout.close();
}
```

```
/*-------------------------------void calcPhase()-------------------------------
Berechne Phasendifferenz. Vergleiche dazu Nulldurchgaenge des Motors und der
Schwingung. Da die Schwingung der Scheibe dem Motor nachfolgt, soll stets die
Differenz in x zwischen einem Nulldurchgang der Scheibe und dem vorherigen
Nulldurchgang des Motors berechnet werden. Falls schon ein Nulldurchgang der
Scheibe vor dem ersten Nulldurchgang des Motors registriert wird, wird dieser
aus dem vector entfernt.-----------------------------------------------------*/

float calcPhase()
{
        if (xZeroCross[0] <= xExitCross[0])
        {
                xZeroCross.erase(xZeroCross.begin());
        }

        vector<float> phase;

        for (int i=0; i<xExitCross.size()-1; i++)
        {
                phase.push_back(xZeroCross[i] - xExitCross[i]);
        }

        float phaseDiff = 0;
        for (int i=0; i<phase.size(); i++)
        {
                phaseDiff += phase[i];
        }

        phaseDiff = phaseDiff / (float)phase.size();

        return phaseDiff;
}



/*-------------------------------void printPhase()------------------------------
Schreibe Zeitdifferenz zwischen den Nulldurchgaengen zusammen mit Informationen
ueber die Schwingung in Datei ("PhaseDiff.txt"). Falls diese schon existiert, als
Anhang. Format: Daempfung Frequenz Zeitdifferenz Amplitude.
-----------------------------------------------------------------------------*/

void printPhase()
{
        ofstream fout;
        fout.open("PhaseDiff.txt", ios::out | ios::app);

        fout << expInfo << "\t" <<  calcPhase() << "␣"
                << yMaxima[yMaxima.size() - 1] << endl;

        fout.close();
}
```