WS19/20, PAP2.1, Versuch 232: Michelson-Interferometer

Praktikanten:
Gerasimov, V., Großmann, J. & Trautmann, G.

Betreuer:
Wachs, D.

Versuchsdurchführung:
19. November, 2019

Inhaltsverzeichnis

1 Einführung

In diesem Versuch wollen wir unter Berücksichtigung von statistischen Fehlern mit Hilfe eines Michelson-Interferometer verschiedene Messungen durchführen:

- Die Wellenlänge von einem grünen Laser messen und mit dessen Herstellerangaben vergleichen
- Den Brechungsindex von Luft für Normalbedingungen bestimmen und und ihn auch mit den Literaturwerten vergleichen.
- Die Kohärenzlänge einer Leuchtdiode messen.

Dafür verwenden wir die Eigenschaften von Interferenzen gleicher Dicke und Interferenzen gleicher Neigung.

2 Versuchsaufbau, Literaturwerte & Vorbereitung

- Michelson-Interferometer
- grüner Laser mit Herstellerangaben der Wellenlänge $\boldsymbol{\lambda = (532 \pm 1) nm}$
- Leuchtdiode
- Küvette mit Vakuumpumpe
- Oszilloskop
- $\boldsymbol{n_{0,Literatur} = 1.00028(1)}$ als Literaturwert für den Brechungsindex von Luft unter Normalbedingungen

3 Messung der Wellenlänge

3.1 Durchführung[1]

Wir verstellen den festen Spiegel so, dass wir etwa 2 bis 3 Interferenzringe auf dem Detektor beobachten. Die Öffnung der Irisblende am Detektor wird auf einen Durchmesser von etwa 1 mm eingestellt. Wir schalten das Oszilloskop ein. An Kanal 1 ist direkt das Detektorsignal angeschlossen. Dieses Signal wird zusätzlich auf den Eingang eines Diskriminators gegeben. Die Elektronik bewirkt, dass das Signal nochmals verstärkt wird und in ein Rechtecksignal umgewandelt wird (Abb.1).

Jeder Impuls entspricht einem Interferenzmaximum. Die Maxima werden mit dem integrierten Zähler gezählt. Bei der Messung der Wellenlänge ist es wichtig, dass tatsächlich jeder Interferenzstreifen detektiert und gezählt wird. Wir verfahren dazu mit dem Wipptaster der Motorsteuerung den beweglichen Spiegel und beobachten dabei das Oszilloskopbild. Bei jedem Maximum des Sinussignals muss der Diskriminator einen Rechteckpuls ausgeben. Falls dies nicht der Fall ist, kann man folgendes optimieren:

- Variierung des Durchmessers der Irisblende am Detektor
- Verstärkung des Detektors (Drehschalter am Detektorgehäuse auf **60** dB bzw. **70** dB stellen).
- Verstärkung des Diskriminators.

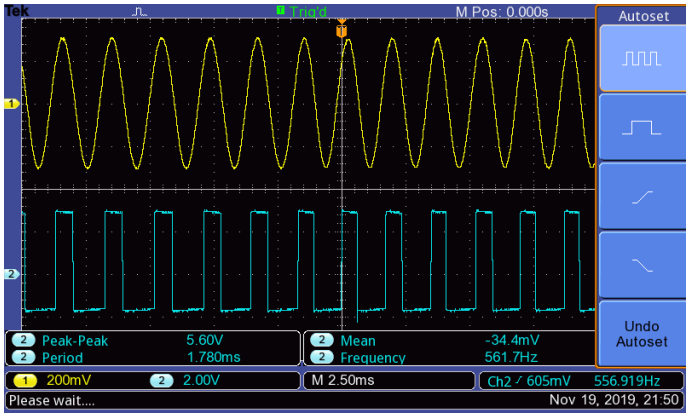

Abbildung 1: Messung der Signale mit einem Oszilloskop

Die Irisblende sollte nicht zu groß eingestellt werden, sonst mitteln wir über das Interferenzmuster. In der Regel reicht es aus die Verstärkung am Diskriminator zu optimieren. Die Verstärkung sollte aber auch nicht zu groß gewählt werden, sonst wird der Aufbau zu empfindlich und anfällig gegen kleinste Erschütterungen. Wir machen uns mit der Messuhr vertraut.

Ein Teilstrich entspricht $\mathbf{1}\ \boldsymbol{\mu m}$,
eine volle Zeigerumdrehung $\mathbf{0.2}\ \boldsymbol{mm} = \mathbf{200}\ \boldsymbol{\mu m}$.

Wir fahren den Spiegel in Richtung Messuhr und stellen die Position auf einen Startwert $\boldsymbol{s_a}$. Beim Einnehmen der Startposition darf nicht die Verfahrrichtung des Motors nicht geändert werden. Wir notieren bei jedeme Messdurchgang den Startwert $\boldsymbol{s_a}$. Ab jetzt sind alle Erschütterungen zu vermeiden. Wir drücken die Resettaste am Zähler und starten den Motor indem wir den rechten oberen Knopf am Motorcontroller vorsichtig drucken. Der Spiegel bewegt sich nun ca. 3 mm und die dabei durchlaufenden Interferenzmaxima werden vom Zähler registriert. Sobald der Spiegel angehalten hat notieren wir sofort die gemessenen Impulse. Danach notieren wir auch die Endposition $\boldsymbol{s_e}$ der Messuhr. Diese Messung ist insgesamt 5 Mal mit jeweils anderen Werten für $\boldsymbol{s_0}$ durchzuführen. Da die Messuhr maximal 5 mm aufnehmen kann, muss die Startposition kleiner als 2 mm sein. Für die Wellenlänge $\boldsymbol{\lambda}$ gilt dann

Tabelle 1: Messung der Wellenlänge

Messung Nr	Startposition $\boldsymbol{s_a}$ $[mm]$	Endposition $\boldsymbol{s_e}$ $[mm]$	Anzahl der Impulse $\boldsymbol{m}$ [1]
1	1.000 ± 0.009	3.957 ± 0.009	11092 ± 2
2	3.957 ± 0.009	3.957 ± 0.009	11092 ± 2
3	1.000 ± 0.009	3.957 ± 0.009	11092 ± 2
4	3.957 ± 0.009	3.957 ± 0.009	11092 ± 2
5	1.000 ± 0.009	3.958 ± 0.009	11092 ± 2

[1] sowohl $\boldsymbol{\Delta s_a}$ als auch $\boldsymbol{\Delta s_e}$ wurdem dem bei der Messuhr ausliegenden Datenblatt entnommen
[2] $\boldsymbol{\Delta m}$ grob abgeschätzt

$$\boldsymbol{\lambda = 2\frac{|s_e - s_a|}{m}} \tag{1}$$

wobei $\boldsymbol{m}$ die gezählten Impulse sind. Die Genauigkeit der Messuhr $\boldsymbol{\Delta s}$ können wir aus dem ausliegenden Datenblatt entnehmen und zu den Messwerten notieren.

[1] Dr. J.Wagner - Physikalisches Anfängerpraktikum - V. 1.1 Stand 1/2018, Versuch 232

3.2 Messergebnisse

Messdaten wurden dem Versuchsprotokoll (19. Novemberr, 2019) entnommen und in Tabelle 1 übertragen.

3.3 Kurvenanpassung mit Python

3.3.1 Source Code & Input

Wir können $\boldsymbol{\lambda}$ nach (1) berechnen:

$$\boldsymbol{\lambda = 2\frac{|s_e - s_a|}{m}} \tag{1}$$

$$\boldsymbol{\Delta\lambda = \lambda\sqrt{\frac{(\Delta s_a)^2 + (\Delta s_e)^2}{(s_a - s_e)^2} + \left(\frac{\Delta m}{m}\right)^2}} \tag{2}$$

Wir gehen davon aus, dass die Wellenlänge konstanmt ist. Daher ist unser funktionales Modell für die Ausgleichungsrechnung wie folgt:

$$\boxed{\boldsymbol{\lambda = konst.}} \tag{3}$$

So sieht unsere Python-Implementierung aus:

Header:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from scipy.signal import hilbert
from scipy.stats import norm
from scipy.stats import chi2
from decimal import Decimal
import csv

def format_e(n):
    a = '%e' % Decimal(n)
    return a.split('e')[0].rstrip('0').rstrip('.')+'e'+a.split('e')[1]

def format_plt(n):
    a = '%e' % Decimal(n)
    return r'${'+a.split('e')[0].rstrip('0').rstrip('.')+'}$'+r'${*10^{'+a.split('e')[1]+'}}$'

def gaussian(x, y, mu, sig):
    return norm.pdf(x, mu, sig)*y
```

Messwerte aus Tabelle 1 in SI Einheiten:

```

s_a = np.array([1.000,3.957,1.000,3.957,1.000]) *1e-3
Fehler_s_a = np.array([0.009,0.009,0.009,0.009,0.009]) *1e-3

s_e = np.array([3.957,1.000,3.957,1.000,3.958]) *1e-3
Fehler_s_e = np.array([0.009,0.009,0.009,0.009,0.009]) *1e-3

m = np.array([11092,11121,11121,11122,11122])
Fehler_m = np.array([2,2,2,2,2])
```

Berechnung von $\boldsymbol{\lambda}$ und $\boldsymbol{\Delta\lambda}$ nach (1) bzw. (2):

```
lamda = 2*np.abs(s_e-s_a)/m
Fehler_lamda = lamda*np.sqrt((np.sqrt(Fehler_s_a**2+Fehler_s_e**2)/(s_a-s_e))**2+(Fehler_m/m)**2)
```

Nummerierung (um es später darstellen zu können):

```
n = np.linspace(1,lamda.size,lamda.size)
Fehler_n = 1e-12
```

Fitfunktion (3) wird deklariert:

```
from scipy import odr

def fit_func(p, x):
    (c) = p
    return x*0+c

model = odr.Model(fit_func)
```

darzustellende Daten werden übergeben:

```
x = n
y = lamda
delta_x = Fehler_n
delta_y = Fehler_lamda
```

Startparameter für Ausgleichungsrechnung werden gesetzt, sodass Lösung konvergiert:

```
para0 = [0]

data = odr.RealData(x, y, sx=delta_x, sy=delta_y)
odr = odr.ODR(data, model, beta0=para0 )
out = odr.run()
```

Endgültige Ausgleichungsparameter und ihre Kovarianzmatrix werden ausgelesen:

```
popt = out.beta
perr = out.sd_beta
```

Angabe welche Sigma-Umgebung der Fitfunktion im Diagramm dargestellt werden soll:

```
nstd = 1

popt_top = popt+nstd*perr
popt_bot = popt-nstd*perr
```

Plot-Umgebung wird angegeben:

```
x_fit = np.linspace(min(x)-(max(x)-min(x))/10, max(x)+3+(max(x)-min(x))/10, 1000)
fit = fit_func(popt, x_fit)
fit_top = fit_func(popt_top, x_fit)
fit_bot = fit_func(popt_bot, x_fit)
```

Diagramm (Abb.3) wird erstellt:

```
y_scale = 1e9
fig, ax = plt.subplots(1, figsize=[6.4 * 1.5, 4.8 * 1.5])
plt.ticklabel_format(axis='both', style='sci', scilimits=(0,3), useMathText=True)
plt.title('Messung der Wellenlänge')
plt.errorbar(x, y*y_scale, yerr=delta_y*y_scale, lw=2, ecolor='k', fmt='none', capsize=8, capthick
    =2, label='Messdaten')
plt.plot(x_fit, fit*y_scale, 'C3--', lw=2, label='Fit')
plt.plot(x_fit, fit*0+532, 'C0--', lw=2, label='Herstellerangaben')
ax.fill_between(x_fit, fit_top*y_scale, fit_bot*y_scale, color='C3', alpha=.25, label=str(nstd)+'$\
    sigma$-Umgebung Messung')
ax.fill_between(x_fit, fit_bot*0+532+1, fit_bot*0+532-1, color='C0', alpha=.25, label='1$\sigma$-
    Umgebung Herstellerangabe')
plt.xlabel('Messung Nr.')
plt.ylabel('Wellenlänge $\lambda$ [nm]')
plt.legend(loc='best')

fig.savefig('figures/232_Fig1.pdf', format='pdf', bbox_inches='tight')
```

Der Chi-Quadrat-Test wird durchgeführt unter Berücksichtigung von $\boldsymbol{\Delta t}$ und $\boldsymbol{\Delta \omega}$. D.h. es wird jeweils der senkrechte/orthogonale Abstand der Messwerte zur Fitfunktion (Abb.2) berechnet und normiert[2]. Die Summe der normierten Abstandsquadrate, der χ^2-Wert, wird reduziert.

```
dof = x.size-popt.size
chisquare = np.sum(((fit_func(popt, x)-y)**2)/(delta_y**2+((fit_func(popt, x+delta_x)-fit_func(popt,
    x-delta_x))/2)**2))
chisquare_red = chisquare/dof
prob = round(1-chi2.cdf(chisquare,dof),2)*100
```

Auslesen der Messergebnisse:

```
lamda_mean = popt[0]
Fehler_lamda_mean = perr[0]
```

Ausgabe der Messergebnisse wird erstellt:

[2] P. T. Boggs and J. E. Rogers, "Orthogonal Distance Regression," in "Statistical analysis of measurement error models and applications: proceedings of the AMS-IMS-SIAM joint summer research conference held June 10-16, 1989," Contemporary Mathematics, vol. 112, pg. 186, 1990.

[3] http://mathworld.wolfram.com/LeastSquaresFitting.html

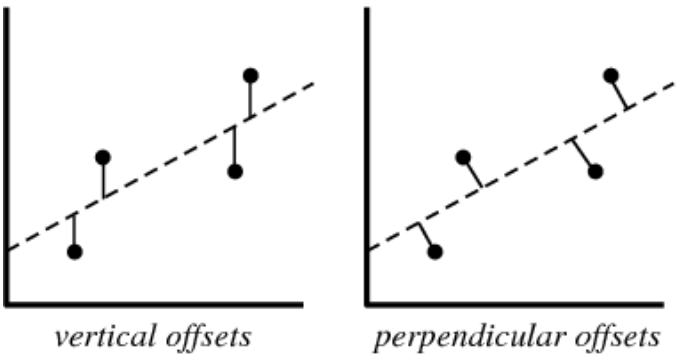

Abbildung 2: Abstände von Messdaten zur Fitfunktion[3]

```
print('Wellenlänge:')
print('lambda [m] =', format_e(lamda_mean), ' +- ', format_e(Fehler_lamda_mean))
print('Chi-Quadrat =', chisquare)
print('Freiheitsgrade =', dof)
print('Chi-Quadrat reduziert =', chisquare_red)
print('Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten =', prob, '%')
```

3.3.2 Output

```
Wellenlänge:
lambda [m] = 5.320805e-07  +-  2.750607e-10
Chi-Quadrat = 0.28801824173344576
Freiheitsgrade = 4
Chi-Quadrat reduziert = 0.07200456043336144
Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten = 99.0 %
```

Wir erfahren also sofort, dass

$$\boldsymbol{\lambda = 532.08(28)\ nm}$$

und als Ergebnis auf unseren Anpassungstest:

$$\boldsymbol{\chi^2_{red} = 7.2 \times 10^{-2}}$$

Die Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten ist

$$\boldsymbol{P \approx 99.0\%}$$

und wir erhalten das Diagramm in Abb.3.

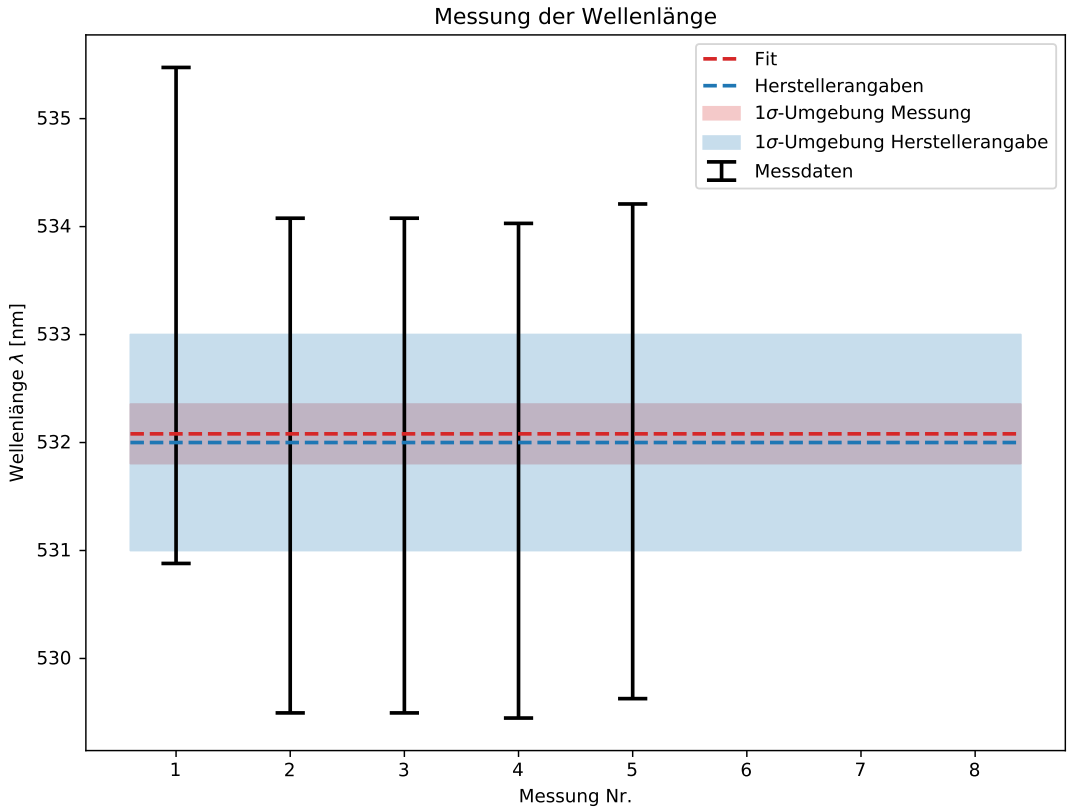

Abbildung 3

3.4 Auswertung

Der Messwert und die Herstellerangabe

$$\boldsymbol{\lambda = 532.08(28)\ nm}$$
$$\boldsymbol{\lambda_{Hersteller} = 532.0(10)\ nm}$$

liegen jeweils in der 1-Sigma-Umgebung von einander. D.h. sie unterscheiden sich nicht signifikant von einander. Der relativ geringe Chi-Quadrat-Wert $\boldsymbol{\chi^2_{red} = 7.2 \times 10^{-2}}$ ist wahrscheinlich einer sehr konservativen Wahl unserer Fehler geschuldet. Der deutlich größere Fehler laut Herstellerangabe ist wahrscheinlich ein willkürlich aufgerundeter Wert auf ganze Nanometer. Es ist zu erwarten das der Hersteller in der Lage wäre die Wellenlänge $\boldsymbol{\lambda}$ genauer zu messen als wir und gegebenenfalls theoretisch zu bestätigen.

4 Messung des Brechungsindex von Luft

4.1 Durchführung[4]

Als Lichtquelle wird weiterhin der grüne Laser benutzt. Den Schirm wird vor dem Detektor platziert und vor dem festen Spiegel auf Position 3 die Küvette. Der feste Spiegel wird so eingestellt, dass auf dem Schirm 2-3 Interferenzringe zu sehen sind. Wir schließen das Nadelventil indem wir den Einstellknopf ganz nach rechts drehen, schalten die Vakuumpumpe ein und öffnen den Absperrhebel solange, bis sich der Druck in der Küvette nicht mehr ändert. Danach können wir den Absperrhebel wieder schließen und die Pumpe ausschalten. Mittels des Nadelventils lässt man in die evakuierte Gaszelle langsam Luft einströmen. Dabei quellen aus dem Zentrum neue Interferenzringe hervor. Wir lesen nach je 5 Interferenzringen den Druck $\boldsymbol{p'}$ am Manometer ab, wobei $\boldsymbol{p = p' + b}$ Wir führen die Messung dreimal durch und notieren zusätzlich die Zimmertemperatur $\boldsymbol{T}$.

Wir berechnen den Brechungsindex von Luft für Normalbedingungen. Ist $\boldsymbol{n_0}$ der Brechungsindex bei Normalbedingungen, dann gilt:

$$\frac{\boldsymbol{n_0 - 1}}{\boldsymbol{n\,(p) - 1}} = \frac{\boldsymbol{p_0 T}}{\boldsymbol{p T_0}} \tag{4}$$

Mit Hilfe von Gleichung (5)

$$\boldsymbol{n\,(\lambda, R, b) - 1 = m \frac{\lambda}{2a} b} \tag{5}$$

wobei $\boldsymbol{b}$ den äußeren Luftdruck und $\boldsymbol{a}$ das Innenmaß der Küvette darstellt. Daraus folgt dann:

$$\boldsymbol{(n_0 - 1) = (n - 1)\,\frac{p_0}{p}\,\frac{T}{T_0} = \frac{\lambda}{2a}\,\frac{m}{p}\,\frac{p_0 T}{T_0}} \tag{6}$$

Die Normalbedingungen sind festgelegt und $\boldsymbol{a}$ ist angegeben[4]als:

$$\boldsymbol{T_0 = 273.15\ K}$$
$$\boldsymbol{p_0 = 1013.25\ hPa}$$
$$\boldsymbol{a = 50.00(5)\ mm}$$

Tabelle 2: des Brechungsindex von Luft

Messreihe Nr	Anzahl der verstrichenen Ringe $\boldsymbol{m}$ [1]	Druck $\boldsymbol{p'}$ $[Torr]$
1	0 ± 2	-743 ± 5
〃	10 ± 2	-670 ± 5
〃	15 ± 2	-595 ± 5
〃	20 ± 2	-515 ± 5
〃	25 ± 2	-440 ± 5
〃	30 ± 2	-365 ± 5
〃	35 ± 2	-290 ± 5
〃	40 ± 2	-200 ± 5
〃	45 ± 2	-135 ± 5
〃	50 ± 2	-55 ± 5
〃	55 ± 2	0 ± 5
2	0 ± 2	-744 ± 5
〃	10 ± 2	-665 ± 5
〃	15 ± 2	-590 ± 5
〃	20 ± 2	-525 ± 5
〃	25 ± 2	-440 ± 5
〃	30 ± 2	-360 ± 5
〃	35 ± 2	$-290, \pm 5$
〃	40 ± 2	-215 ± 5
〃	45 ± 2	-140 ± 5
〃	50 ± 2	-60 ± 5
3	0 ± 2	-745 ± 5
〃	10 ± 2	-670 ± 5
〃	15 ± 2	-595 ± 5
〃	20 ± 2	-515 ± 5
〃	25 ± 2	-440 ± 5
〃	30 ± 2	-365 ± 5
〃	35 ± 2	-290 ± 5
〃	40 ± 2	-213 ± 5
〃	45 ± 2	-155 ± 5
〃	50 ± 2	-60 ± 5

[1] $\boldsymbol{\Delta p}$ wurden als 0,6% vom Skalenendwert (800 Torr) abgeschätzt[4]

[2] $\boldsymbol{\Delta m}$ grob abgeschätzt

4.2 Messergebnisse

Messdaten wurden dem Versuchsprotokoll (19. Novemberr, 2019) entnommen und in Tabelle 2 übertragen.
Die Raumtemperatur ist $\boldsymbol{T = 23.8(2)\ °C}$ mit dem Fehler $\boldsymbol{\Delta T}$ abgeschätzt aus der Inhomogenität des Anzeigewerts am Thermometer während der Versuchszeit.

4.3 Kurvenanpassung mit Python, Schritt 1

4.3.1 Source Code & Input

Wir gehen davon aus, dass die Präzessionsdauer proportional zur Eigenkreisfrequenz zunimmt. Daher ist unser funktionales Modell für die Ausgleichungsrechnung wie folgt:

$$\boxed{\boldsymbol{p' = mS + konst.}} \tag{7}$$

[4]Dr. J.Wagner - Physikalisches Anfängerpraktikum - V. 1.1 Stand 1/2018, Versuch 232

So sieht die Fortführung unserer Python-Implementierung aus:

Messwerte aus Tabelle 2 in SI Einheiten (wobei $\boldsymbol{m}$ für die einzelnen Messreihen relative verschoben wurden, um sie besseer darstellen zu können):

```
m_1 = np.array([0,10,15,20,25,30,35,40,45,50,55])
Fehler_m_1 = np.array([2,2,2,2,2,2,2,2,2,2,2])

m_2 = np.array([0,10,15,20,25,30,35,40,45,50]) +10
Fehler_m_2 = np.array([2,2,2,2,2,2,2,2,2,2])

m_3 = np.array([0,10,15,20,25,30,35,40,45,50]) +20
Fehler_m_3 = np.array([2,2,2,2,2,2,2,2,2,2])

p_1 = np.array([-743,-670,-595,-515,-440,-365,-290,-200,-135,-55,0]) *(101325/760)
Fehler_p_1 = np.array([5,5,5,5,5,5,5,5,5,5,5]) *(101325/760)

p_2 = np.array([-744,-665,-590,-525,-440,-360,-290,-215,-140,-60]) *(101325/760)
Fehler_p_2 = np.array([5,5,5,5,5,5,5,5,5,5]) *(101325/760)

p_3 = np.array([-745,-670,-595,-515,-440,-365,-290,-213,-155,-60]) *(101325/760)
Fehler_p_3 = np.array([5,5,5,5,5,5,5,5,5,5]) *(101325/760)
```

Fitfunktion (7) wird deklariert:

```
from scipy import odr

def fit_func(p, x):
    (s, c) = p
    return s*x+c

model = odr.Model(fit_func)
```

darzustellende Daten werden übergeben:

```
x_1 = p_1
y_1 = m_1
delta_x_1 = Fehler_p_1
delta_y_1 = Fehler_m_1

x_2 = p_2
y_2 = m_2
delta_x_2 = Fehler_p_2
delta_y_2 = Fehler_m_2

x_3 = p_3
y_3 = m_3
delta_x_3 = Fehler_p_3
delta_y_3 = Fehler_m_3
```

Startparameter für Ausgleichungsrechnung werden gesetzt, sodass Lösung konvergiert:

```
para0 = [0, 0]

data_1 = odr.RealData(x_1, y_1, sx=delta_x_1, sy=delta_y_1)
odr_1 = odr.ODR(data_1, model, beta0=para0 )
out_1 = odr_1.run()

data_2 = odr.RealData(x_2, y_2, sx=delta_x_2, sy=delta_y_2)
odr_2 = odr.ODR(data_2, model, beta0=para0 )
out_2 = odr_2.run()

data_3 = odr.RealData(x_3, y_3, sx=delta_x_3, sy=delta_y_3)
odr_3 = odr.ODR(data_3, model, beta0=para0 )
out_3 = odr_3.run()
```

Endgültige Ausgleichungsparameter und ihre Kovarianzmatrix werden ausgelesen:

```
popt_1 = out_1.beta
perr_1 = out_1.sd_beta

popt_2 = out_2.beta
perr_2 = out_2.sd_beta

popt_3 = out_3.beta
perr_3 = out_3.sd_beta
```

Angabe welche Sigma-Umgebung der Fitfunktion im Diagramm dargestellt werden soll:

```
nstd = 1

popt_top_1 = popt_1+nstd*perr_1
popt_bot_1 = popt_1-nstd*perr_1

popt_top_2 = popt_2+nstd*perr_2
popt_bot_2 = popt_2-nstd*perr_2

popt_top_3 = popt_3+nstd*perr_3
popt_bot_3 = popt_3-nstd*perr_3
```

Plot-Umgebung wird angegeben:

```
x_fit = np.linspace(min(min(x_1),min(x_2),min(x_3))*1.1, min(min(x_1),min(x_2),min(x_3))*(-0.1),
    1000)

fit_1 = fit_func(popt_1, x_fit)
fit_top_1 = fit_func(popt_top_1, x_fit)
fit_bot_1 = fit_func(popt_bot_1, x_fit)

fit_2 = fit_func(popt_2, x_fit)
fit_top_2 = fit_func(popt_top_2, x_fit)
fit_bot_2 = fit_func(popt_bot_2, x_fit)

fit_3 = fit_func(popt_3, x_fit)
fit_top_3 = fit_func(popt_top_3, x_fit)
fit_bot_3 = fit_func(popt_bot_3, x_fit)
```

Diagramm (Abb.4) wird erstellt:

```
fig, ax = plt.subplots(1, figsize=[6.4 *1.5, 4.8])
plt.ticklabel_format(axis='both', style='sci', scilimits=(0,3), useMathText=True)
plt.errorbar(x_1, y_1, yerr=delta_y_1, xerr=delta_x_1, lw=1, ecolor='k', fmt='none', capsize=1,
    label='Messreihe 1')
plt.errorbar(x_2, y_2, yerr=delta_y_2, xerr=delta_x_2, lw=1, ecolor='k', fmt='none', capsize=1,
    label='Messreihe 2')
plt.errorbar(x_3, y_3, yerr=delta_y_3, xerr=delta_x_3, lw=1, ecolor='k', fmt='none', capsize=1,
    label='Messreihe 3')
plt.title('Messung der Steigung $S_i$ = $\Delta m_i / p$ pro Messreihe')
plt.grid(True)
plt.xlabel('Druck $p+p_0$ [Pa]')
plt.ylabel('$m$')
plt.plot(x_fit, fit_1, color='C0', lw=1, label='Fit  1')
plt.plot(x_fit, fit_2, color='C2', lw=1, label='Fit  2')
plt.plot(x_fit, fit_3, color='C3', lw=1, label='Fit  3')
ax.fill_between(x_fit, fit_top_1, fit_bot_1, color='C0', alpha=.25, label=str(nstd)+r'$\sigma$'+'-
    Umgebung 1')
ax.fill_between(x_fit, fit_top_2, fit_bot_2, color='C2', alpha=.25, label=str(nstd)+r'$\sigma$'+'-
    Umgebung 2')
ax.fill_between(x_fit, fit_top_3, fit_bot_3, color='C3', alpha=.25, label=str(nstd)+r'$\sigma$'+'-
    Umgebung 3')
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.savefig('figures/232_Fig2.pdf', format='pdf', bbox_inches='tight')
```

Der Chi-Quadrat-Test wird durchgeführt unter Berücksichtigung von $\boldsymbol{\Delta\omega_F}$ und $\boldsymbol{\Delta T_P}$. D.h. es wird jeweils der senkrechte/orthogonale Abstand der Messwerte zur Fitfunktion (Abb.2) berechnet und normiert[5].

```
dof_1 = x_1.size-popt.size
chisquare_1 = out_1.sum_square
chisquare_red_1 = chisquare_1/dof_1
prob_1 = round(1-chi2.cdf(chisquare_1,dof_1),2)*100

dof_2 = x_2.size-popt.size
chisquare_2 = out_2.sum_square
chisquare_red_2 = chisquare_2/dof_2
prob_2 = round(1-chi2.cdf(chisquare_2,dof_2),2)*100

dof_3 = x_3.size-popt.size
```

[5] P. T. Boggs and J. E. Rogers, "Orthogonal Distance Regression," in "Statistical analysis of measurement error models and applications: proceedings of the AMS-IMS-SIAM joint summer research conference held June 10-16, 1989," Contemporary Mathematics, vol. 112, pg. 186, 1990.

```
chisquare_3 = out_3.sum_square
chisquare_red_3 = chisquare_3/dof_3
prob_3 = round(1-chi2.cdf(chisquare_3,dof_3),2)*100
```

Auslesen der Messergebnisse:

```
S_1 = popt_1[0]
Fehler_S_1 = perr_1[0]
S_2 = popt_2[0]
Fehler_S_2 = perr_2[0]
S_3 = popt_3[0]
Fehler_S_3 = perr_3[0]
```

Ausgabe der Messergebnisse wird erstellt:

```
print('Steigungen: ')
print('S_1 [1/Pa] =', format_e(S_1), ' +- ', format_e(Fehler_S_1))
print('Chi-Quadrat =', chisquare_1)
print('Freiheitsgrade =', dof_1)
print('Chi-Quadrat reduziert =', chisquare_red_1)
print('Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten =', prob_1, '%')
print('\n')
print('S_2 [1/Pa] =', format_e(S_2), ' +- ', format_e(Fehler_S_2))
print('Chi-Quadrat =', chisquare_2)
print('Freiheitsgrade =', dof_2)
print('Chi-Quadrat reduziert =', chisquare_red_2)
print('Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten =', prob_2, '%')
print('\n')
print('S_3 [1/Pa] =', format_e(S_3), ' +- ', format_e(Fehler_S_3))
print('Chi-Quadrat =', chisquare_3)
print('Freiheitsgrade =', dof_3)
print('Chi-Quadrat reduziert =', chisquare_red_3)
print('Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten =', prob_3, '%')
```

4.3.2 Output

```
Steigungen:
S_1 [1/Pa] = 5.171618e-04  +-  1.381261e-05
Chi-Quadrat = 4.683194878131682
Freiheitsgrade = 10
Chi-Quadrat reduziert = 0.46831948781316823
Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten = 91.0 %

S_2 [1/Pa] = 5.224287e-04  +-  1.573934e-05
Chi-Quadrat = 4.0422956700548145
Freiheitsgrade = 9
Chi-Quadrat reduziert = 0.44914396333942386
Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten = 91.0 %

S_3 [1/Pa] = 5.248941e-04  +-  1.589543e-05
Chi-Quadrat = 4.082827181272153
Freiheitsgrade = 9
Chi-Quadrat reduziert = 0.45364746458579475
Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten = 91.0 %
```

Wir erfahren also sofort, dass

$$\boldsymbol{S_1 = 5.17(14) \times 10^{-4}\, Pa^{-1}}$$
$$\boldsymbol{S_2 = 5.22(16) \times 10^{-4}\, Pa^{-1}}$$
$$\boldsymbol{S_3 = 5.25(16) \times 10^{-4}\, Pa^{-1}}$$

und als Ergebnis auf unseren Anpassungstests:

$$\boldsymbol{\chi^2_{red,1} = 4.7 \times 10^{-1}}$$
$$\boldsymbol{\chi^2_{red,2} = 4.5 \times 10^{-1}}$$
$$\boldsymbol{\chi^2_{red,3} = 4.5 \times 10^{-1}}$$

Die Wahrscheinlichkeiten ein größeres oder gleiches Chi-Quadrat zu erhalten sind jeweils

$$\boldsymbol{P_1 \approx 91.0\%}$$
$$\boldsymbol{P_2 \approx 91.0\%}$$
$$\boldsymbol{P_3 \approx 91.0\%}$$

(Rechnung mehrmals überprüft. Alle 3 Werte stimmen wirklich auf zwei Stellen überein)
und wir erhalten das Diagramm in Abb.4.

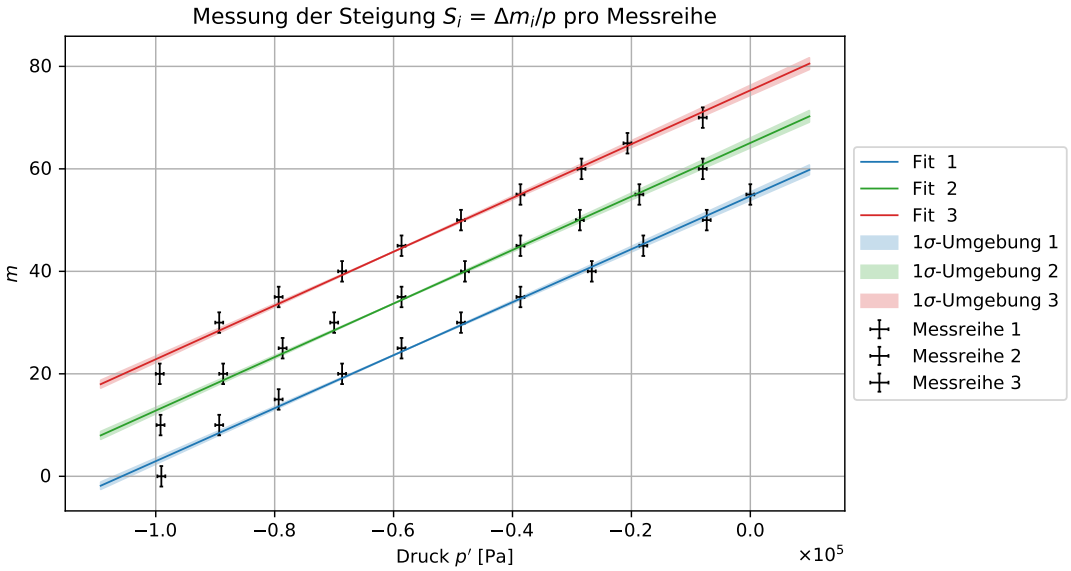

Abbildung 4

4.4 Kurvenanpassung mit Python, Schritt 2

4.4.1 Source Code & Input

Wir gehen davon aus, dass der Brechungsindex konstant ist. Daher ist unser funktionales Modell für die Ausgleichungsrechnung wie folgt:

$$\boxed{\boldsymbol{n_0 = konst.}} \tag{8}$$

Für diesen Schritt, benutzen wir die Angaben aus "Versuchsaufbau, Literaturwerte & Vorbereitung"

$$\boldsymbol{T_0 = 273.15\ K}$$
$$\boldsymbol{p_0 = 1013.25\ hPa}$$
$$\boldsymbol{a = 50.00(5)\ mm}$$

unser Ergebnis für $\boldsymbol{\lambda}$ aus der vorherigen Messung

$$\boldsymbol{\lambda = 532.08(28)\ nm}$$

und die Messergebnisse aus Schritt 1 um den Brechungsindex nach (6) zu berechnen:

$$\boldsymbol{n_0 = 1 + \frac{\lambda}{2a}\frac{\Delta m}{p}\frac{p_0 T}{T_0} = 1 + S\frac{\lambda}{2a}\frac{p_0 T}{T_0}} \tag{9}$$

$$\boldsymbol{\Delta n_0 = \frac{\lambda S p_0 T}{2a T_0}\sqrt{\left(\frac{\Delta\lambda}{\lambda}\right)^2 + \left(\frac{\Delta S}{S}\right)^2 + \left(\frac{\Delta T}{T}\right)^2 + \left(\frac{\Delta a}{a}\right)^2}} \tag{10}$$

So sieht die Fortführung unserer Python-Implementierung aus:

Messwerte der Steigung S aus Schritt 1 in SI Einheiten:

```
S = np.array([S_1,S_2,S_3])
Fehler_S = np.array([Fehler_S_1,Fehler_S_2,Fehler_S_3])
```

Raumtemperatur $\boldsymbol{T}$ in SI Einheiten:

```
T = 23.8 +273.15
Fehler_T = 0.2
```

Normalbedingungen in SI Einheiten:

```
p_0 = 101325
T_0 = 273.15
```

Innenmaß der Küvette $\boldsymbol{a}$ in SI Einheiten:

```
a = 50 *1e-3
Fehler_a = 0.05 *1e-3
```

Berechnung des Brechungsindex $\boldsymbol{n_0}$ und $\boldsymbol{\Delta n_0}$ nach (9) bzw. (10):

```
n_0 = 1+lamda_mean*S*p_0*T/(2*a*T_0)
Fehler_n_0 = lamda_mean*S*p_0*T/(2*a*T_0)*np.sqrt((Fehler_lamda_mean/lamda_mean)**2+(Fehler_S/S)**2
                                                  +(Fehler_T/T)**2+(Fehler_a/a)**2)
```

Nummerierung (um es später darstellen zu können):

```
N = np.linspace(1,S.size,S.size)
Fehler_N = 1e-12
```

Fitfunktion (8) wird deklariert:

```
from scipy import odr

def fit_func(p, x):
    (c) = p # c: Konstante
    return x*0+c

model = odr.Model(fit_func)
```

darzustellende Daten werden übergeben:

```
x = N
y = n_0
delta_x = Fehler_N
delta_y = Fehler_n_0
```

Endgültige Ausgleichungsparameter und ihre Kovarianzmatrix werden ausgelesen:

```
para0 = [0]

data = odr.RealData(x, y, sx=delta_x, sy=delta_y)
odr = odr.ODR(data, model, beta0=para0 )
out = odr.run()
```

Endgültige Ausgleichungsparameter und ihre Kovarianzmatrix werden ausgelesen:

```
popt = out.beta
perr = out.sd_beta
```

Angabe welche Sigma-Umgebung der Fitfunktion im Diagramm dargestellt werden soll:

```
nstd = 1 # um n-Sigma-Umgebung im Diagramm zu zeichnen

popt_top = popt+nstd*perr
popt_bot = popt-nstd*perr
```

Plot-Umgebung wird angegeben:

```
x_fit = np.linspace(min(x)-(max(x)-min(x))/10, max(x)+3+(max(x)-min(x))/10, 1000)
fit = fit_func(popt, x_fit)
fit_top = fit_func(popt_top, x_fit)
fit_bot = fit_func(popt_bot, x_fit)
```

Diagramm (Abb.5) wird erstellt:

```
fig, ax = plt.subplots(1, figsize=[6.4 * 1.5, 4.8 * 1.5])
plt.ticklabel_format(axis='both', style='sci', scilimits=(0,3), useMathText=True)
plt.title('Messung des Brechungsindex')
plt.errorbar(x, y, yerr=delta_y, lw=2, ecolor='k', fmt='none', capsize=8, capthick=2, label='
    Ergebnisse der Messreihen')
plt.plot(x_fit, fit, 'C3--', lw=2, label='Fit')
ax.fill_between(x_fit, fit_top, fit_bot, color='C3', alpha=.25, label=str(nstd)+'$\sigma$-Umgebung
    Fit')
plt.plot(x_fit, fit*0+1.00028, 'C0--', lw=2, label='Literaturwert')
plt.xlabel('Messreihe Nr.')
plt.ylabel('Brechungsindex $n_0$')
plt.legend(loc='best')

fig.savefig('figures/232_Fig3.pdf', format='pdf', bbox_inches='tight')

#Chi-Quadrat orthogonal
dof = x.size-popt.size
chisquare = np.sum(((fit_func(popt, x)-y)**2)/(delta_y**2+((fit_func(popt, x+delta_x)-fit_func(popt,
    x-delta_x))/2)**2))
chisquare_red = chisquare/dof
prob = round(1-chi2.cdf(chisquare,dof),2)*100

#Auswertung
n_0_mean = popt[0]
Fehler_n_0_mean = perr[0]

#Ausgabe
print('Brechungsindex:')
print('n_0 =', n_0_mean, ' +- ', format_e(Fehler_n_0_mean))
print('Chi-Quadrat =', chisquare)
print('Freiheitsgrade =', dof)
print('Chi-Quadrat reduziert =', chisquare_red)
print('Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten =', prob, '%')
```

4.4.2 Output

```
Brechungsindex:
n_0 = 1.000305408647939  +-  1.373493e-06
Chi-Quadrat = 0.14506966547849148
Freiheitsgrade = 2
Chi-Quadrat reduziert = 0.07253483273924574
Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten = 93.0 %
```

Wir erfahren also sofort, dass

$$\boldsymbol{n_0 = 1.0003054(14)}$$

und als Ergebnis auf unseren Anpassungstest:

$$\boldsymbol{\chi^2_{red} = 7.3 \times 10^{-2}}$$

Die Wahrscheinlichkeit ein größeres oder gleiches Chi-Quadrat zu erhalten ist

$$\boldsymbol{P \approx 93.0\%}$$

und wir erhalten das Diagramm in Abb.5.

4.5 Auswertung

Die gemessenen Wert für $\boldsymbol{n_0}$ passt zwar Sinn und passen zu unserem theoretischen Modellen (7) und (8), da keine signifikant abweichenden Chi Quadratwerte auftrafen und Wahrscheinlichkeiten ein größeres oder gleiches Chi-Quadrat zu erhalten waren:

$$\boldsymbol{P_1 \approx 91.0\%}$$
$$\boldsymbol{P_2 \approx 91.0\%}$$
$$\boldsymbol{P_3 \approx 91.0\%}$$
$$\boldsymbol{P \approx 93.0\%}$$

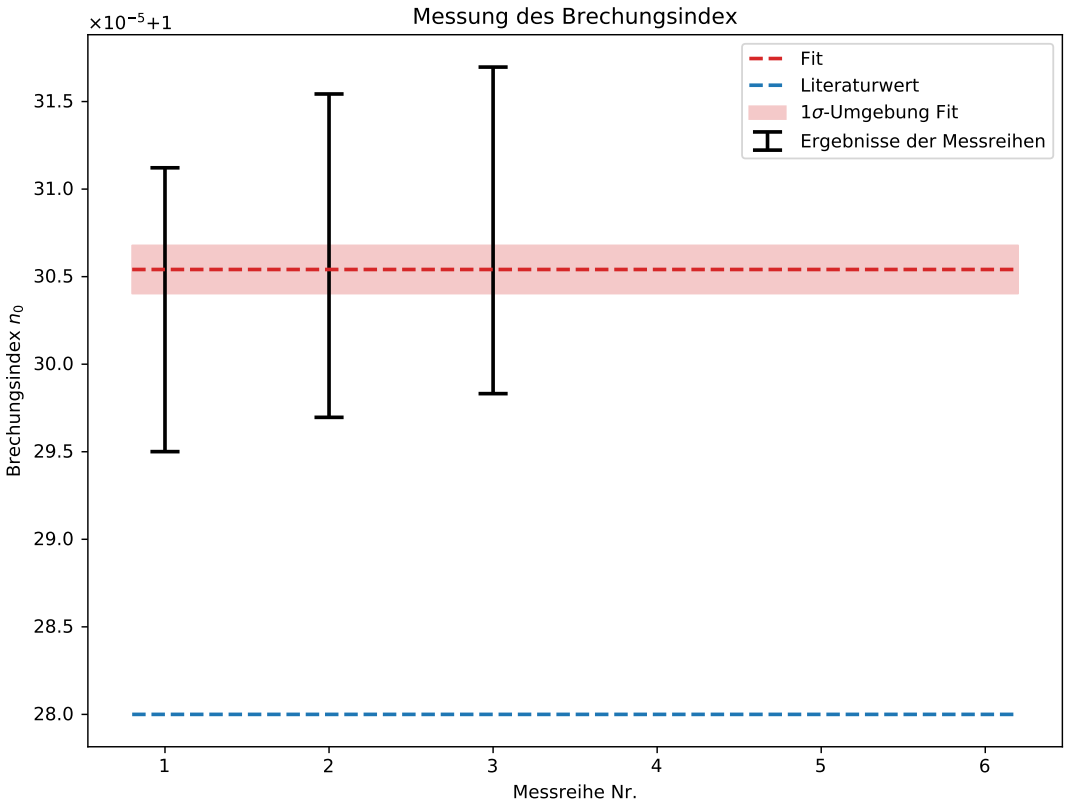

Abbildung 5

Daran lässt sich lässt sich aber vermuten, dass unsere abgeschätzten Messfehler $\boldsymbol{\Delta p}$ und $\boldsymbol{\Delta m}$ zu groß waren. Auf signifikante systematische Fehler lassen die Werte aber nicht schließen. Der Vergleich von Messergebnis und Literaturwert lässt dagegen deutlich werden, dass bei uns an irgendeiner Stelle doch systematische Fehler aufgetreten sein müssen:

$$\boldsymbol{n_0 = 1.0003054(14)}$$
$$\boldsymbol{n_{0,Literatur} = 1.00028(1)}$$

Da der Literaturwert nur sehr grob angegeben wurde, handelt es sich nur um eine $\boldsymbol{2.5\sigma}$-signifikante Differenz.

$$\boldsymbol{\frac{n_{0,Literatur} - n_0}{\sqrt{(\Delta n_{0,Literatur})^2 + (\Delta n_0)^2}} = 2.5}$$

Damit ist die Wahrscheinlichkeit, dass diese Differenz hauptsächlich statistisch bedingt ist nur

$$\boldsymbol{1 - erf\left(\frac{2.5}{\sqrt{2}}\right) \approx 1.2\%}$$

Stellen an denen die dafür verantwortlichen statistische Fehler auftreten können, können im Versuchsaufbau aufgetreten sein (z.B. Volumen der Küvette), menschlicher Natur sein (grobes Verzählen, falsche Ablesung der Messwerte) oder physikalische Abhängigkeiten sein, die wir einfach nur vernachlässigt haben (z.B. Luftfeuchtigkeit). Versuch wurde nicht ausführlich genug wiederholt um darüber im Nachhinein eine detaillierte Aussage treffen zu können.

5 Messung der Kohärenzlänge einer Leuchtdiode

5.1 Durchführung[6]

Wir entfernen die Küvette aus dem Strahlengang und verfahren den beweglichen Spiegel in den Bereich der Weißlichtposition. Der ungefähre Wert ist auf dem Interferometer angegeben. Man stellt den festen Spiegel so ein, dass wir sehr große Interferenzstrukturen zu sehen sind, d.h. möglichst nur einen Ring. Anstatt des Lasers bauen wir nun

[6] Dr. J.Wagner - Physikalisches Anfängerpraktikum - V. 1.1 Stand 1/2018, Versuch 232

die Leuchtdiode ein, entfernen die $-50\,mm$ Linse vor dem Detektor und richten die Leuchtdiode so aus, dass wir den Schatten des Strahlteilers zentrisch auf dem Detektor sehen. Die Öffnung der Irisblende am Detektor sollte etwa $1-2\,mm$ sein. Danach stellen wir die Verstärkung des Detektors auf das Maximum ein ($70\,db$).

Da die Kohärenzlänge der Leuchtdiode sehr klein ist, sind Interferenzen nur über einen sehr kleinen Verfahrweg beobachtbar. Um diese mit dem Oszilloskop darzustellen, muss das Oszilloskop im Single-Modus betrieben werden. In diesem Modus „wartet" das Oszilloskop auf ein Signal einer gewissen Größe, welches wir mit dem Triggerlevel einstellen können. Sobald ein Signal anliegt welches den Level übersteigt, beginnt das Oszilloskop mit der Aufzeichnung und speichert den Signalverlauf. Eine Anleitung zur Einstellung des Single- Modus liegt aus. Wir stellen die y-Ablenkung auf $50\,mV$ und legen den Triggerlevel gerade etwas über den Rauschpegel, so dass das Oszilloskop im Single- Modus nicht auslöst.

Das Oszilloskop stellen wir scharf und verfahren den beweglichen Spiegel in Richtung Strahlteiler. Dazu den Wippschalter des Motorcontrollers ganz nach oben drucken und halten. Sobald wir auf die Weißlichtposition kommen löst das Oszilloskop aus und speichert das Signal. Dann lassen wir sofort den Wippschalter los und stellen den Schirm vor den Detektor. Den Spiegel sehr langsam fahren wir in die entgegengesetzte Richtung bis Interferenzen sichtbar werden.

Danach entfernen wir den Schirm und wiederholen die Messung mit optimierten Oszilloskopeinstellungen. Z.B. entfernen wir eine Umdrehung der Messuhr von der Weißlichtposition und fahren dann mit maximaler Geschwindigkeit zurück durch die Weißlichtposition (Wippschalter ganz durchdrücken). Den Signalverlauf speichern wir auf einen USB- Stick.

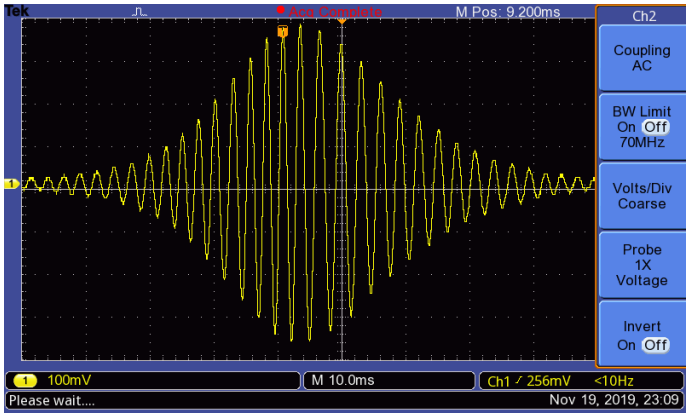

Abbildung 6: Signalverlauf am Oszilloskop zu den Interferenzstrukturen

5.2 Messergebnisse

Messdaten wurden unter der CSV-Datei
`data\Messung2\Messung2_CH1.csv`
(19. Novemberr, 2019) und als BMP-Bild in Abbildung 6 gespeichert.

5.3 Auswertung und Darstellung mit Python

5.3.1 Source Code & Input

Wir drücken t über den entsprechenden Verfahrensweg x aus:

$$x = tv \tag{11}$$

Die Fehler der Werte Δv, Δt und Δx sehen wir in diesem Versuch als vernachlässigbar klein an.
So sieht die Fortführung unserer Python-Implementierung aus:

Verfahrensgeschwindigkeit v in SI Einheiten:

```
v_Verfahren = 0.1 *1e-3
```

Messwerte aus CSV-Datei in SI Einheiten:

```
t = np.array([])
V = np.array([])
with open('data\Messung2\Messung2_CH1.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        t = np.append(t,float(row[3]))
        V = np.append(V,float(row[4]))
```

Berechnung des Verfahrensweges x nach (11):

```
x = t*v_Verfahren
```

Signalverarbeitung zur Erzeugung einer Einhüllenden:

```
analytic_signal = hilbert(V)
amplitude_envelope = np.abs(analytic_signal)
```

Anpassung der Einhüllenden an eine Normalverteilung über die Zeit:

```
fitParams, fitCovariances = curve_fit(gaussian, t, amplitude_envelope)
```

Auslesen der Messergebnisse und Umwandlung der Abhängigkeit von Zeit zur Abhängigkeit vom Weg:

```
mu = fitParams[1]*v_Verfahren
sigma = fitParams[2]*v_Verfahren
counts = t.size
```

Diagramm (Abb.7) wird erstellt:

```
fig, ax = plt.subplots(1, figsize=[6.4 *1.5, 4.8])
plt.ticklabel_format(axis='both', style='sci', scilimits=(0,3), useMathText=True)
plt.plot(x, V, lw=1, color='C3', marker='', alpha=.50, label='Signal')
plt.plot(x,gaussian(t,*fitParams),  lw=1, color='C0', label='Normalverteilung:\n'
         +r'${\mu}$'+' = '+str(format_plt(mu))+' m\n'
         +r'${\sigma}$'+' = '+str(format_plt(sigma))+' m\n'
         +r'$FWHM = $'+format_plt(sigma*2.355)+' m')
plt.title('Messung der Kohärenzlänge einer Leuchtdiode')
plt.ylabel('Spannung U [V]')
plt.xlabel('Verfahrensweg x [m]')
plt.grid(True)
plt.legend(loc='best')

fig.savefig('figures/232_Fig4.pdf', format='pdf', bbox_inches='tight')
```

Ausgabe der Messergebnisse wird erstellt:

```
print('Anzahl der Messungen = ', counts)
print('Normalverteilung:\n'
         +'mu = '+str(format_e(mu))+' m\n'
         +'sigma = '+str(format_e(sigma))+' m\n'
         +'FWHM = '+format_e(sigma*2.355)+' m')
```

5.3.2 Output

```
Anzahl der Messungen =  2500
Normalverteilung:
mu = 3.973149e-07 m
sigma = 1.392349e-06 m
FWHM = 3.278981e-06 m
```

Wir erfahren also sofort, dass die Einhüllende des Signalverlaufs, also der räumlichen Interferenzstruktur ungefähr einer Normalverteilung folgt, mit den folgenden Parametern:

$$\boldsymbol{\mu = 3.973149 \times 10^{-7} m}$$
$$\boldsymbol{\sigma = 1.392349 \times 10^{-6} m}$$
$$\boldsymbol{FWHM = 3.27898 \times 10^{-6} m}$$

und ein Bild der angepassten Normalverteilung und der Interferenzstrukturen in Abbildung 7.

5.4 Auswertung

Die Kohärenzlänge $\boldsymbol{l_c}$ der Leuchtdiode entspricht genau der bestimmten Halbwertsbreite $\boldsymbol{FWHM}$ (eng. Abkürzung für: Full Width at Half Maximum):

$$\boldsymbol{l_c = FWHM = 3.27898\,\mu m}$$

Weil wir zuvor uns dazu entschieden haben keine Fehler bei diesem Teilversuch zu betrachten, haben wir auch keinen Fehler $\boldsymbol{\Delta l_c}$ bestimmt.

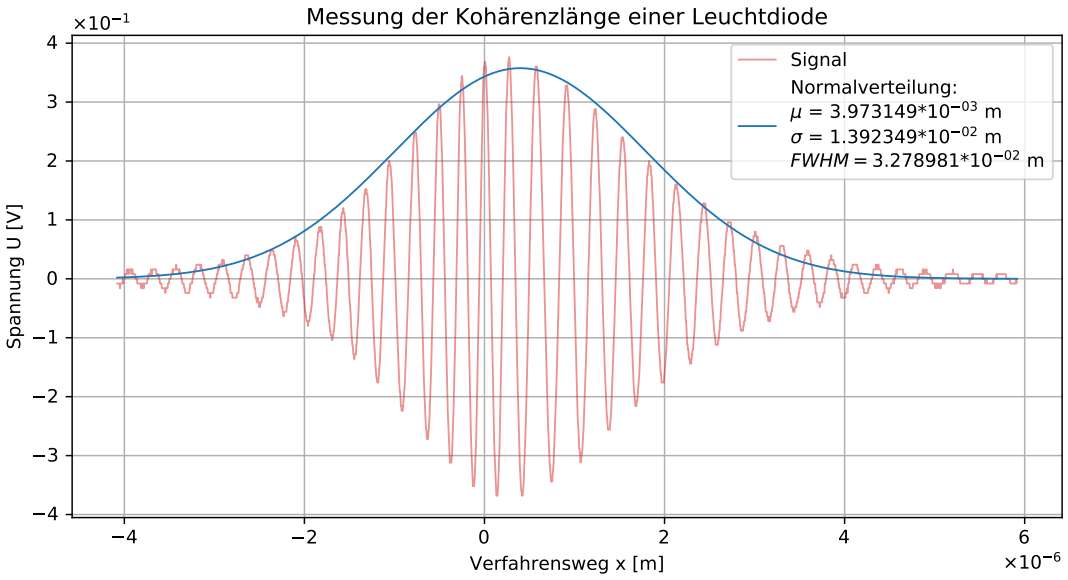

Abbildung 7

6 Fazit

- Die Wellenlänge von einem grünen Laser wurde auf bestimmt und stimmt mit den Herstellerangaben (Differenz kleiner als $\mathbf{1\sigma}$).

$$\boldsymbol{\lambda = 532.08(28)\ nm}$$
$$\boldsymbol{\lambda_{Hersteller} = 532.0(10)\ nm}$$

- Der Brechungsindex von Luft für Normalbedingungen wurde bestimmt und und mit den Literaturwerten verglichen. Die Differenz zwischen ihnen beträgt 2.5 Sigma. Unbeachtete systematische Fehler, die signifikant das Ergebnis verändert haben, sind zu vermuten (siehe Auswertung).

$$\boldsymbol{n_0 = 1.0003054(14)}$$
$$\boldsymbol{n_{0,Literatur} = 1.00028(1)}$$

- Die Kohärenzlänge der uns gegebenen Leuchtdiode wurde auf ungefähr $\boldsymbol{l_c = 3.28\ \mu m}$ bestimmt. Größenordnungsmäßig liegt das unter gewöhnlichem Feinstaub.