

Di02-PW5

November 4, 2025

1 PW5 – Wellenoptik

1.1 Experiment 1 – Beugung am Einzelspalt

1.1.1 Grundlagen

Bei der **Fraunhofer-Beugung am Einzelspalt** wird kohärentes Licht (z. B. eines Lasers) auf einen schmalen Spalt der Breite b gerichtet.

Hinter dem Spalt überlagern sich die Elementarwellen nach dem **Huygens-Fresnel-Prinzip**, wodurch ein charakteristisches Intensitätsmuster aus hellen und dunklen Streifen entsteht.

Die **Bedingung für die Minima** der Intensitätsverteilung lautet:

$$b \sin(\alpha_{\min,n}) = n\lambda, \quad n = 1, 2, 3, \dots$$

Für kleine Beugungswinkel gilt $\sin(\alpha) \approx \alpha$, daher folgt:

$$b \alpha_{\min,n} \approx n\lambda$$

Die Maxima liegen zwischen den Minima bei:

$$b \sin(\alpha_{\max,n}) = \frac{2n+1}{2} \lambda$$

Die Intensitätsverteilung ergibt sich aus:

$$I(\alpha) = I_0 \left(\frac{\sin(\beta)}{\beta} \right)^2, \quad \text{mit} \quad \beta = \frac{\pi b \sin(\alpha)}{\lambda}.$$

1.1.2 Durchführung

Ein Diodenlaser mit Wellenlänge $\lambda = 635 \text{ nm}$ beleuchtet einen Einzelspalt.

Das Beugungsbild wird auf einem Schirm beobachtet und die Abstände der Minima symmetrisch zur zentralen Maxima gemessen.

Zur Vermeidung von Nullpunktfehlern werden die Abstände zwischen den Minima gleicher Ordnung auf beiden Seiten gemessen und halbiert.

Aus dem Abstand y_n des n -ten Minimums und der Entfernung L zwischen Spalt und Schirm ergibt sich der Winkel:

$$\alpha_n \approx \tan^{-1}\left(\frac{y_n}{L}\right) \approx \frac{y_n}{L}.$$

Ein Diagramm α_n gegen n liefert durch lineare Regression die Steigung λ/b .

1.1.3 Auswertung

Die lineare Regression der gemessenen Daten ergibt:

$$\alpha_n = \frac{\lambda}{b} n.$$

Damit folgt für die Spaltbreite:

$$b = \frac{\lambda}{\text{Steigung}}.$$

Zur Qualität der Regression wird das Bestimmtheitsmaß R^2 angegeben.

Die Unsicherheit von b ergibt sich aus der Standardabweichung der Regressionsparameter.

```
[27]: # --- Importe ---
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# --- Hilfsfunktion: schoene Ausgabe im Format "x +/- dx" ---
def pm(val, err):
    if err == 0 or not np.isfinite(err):
        return f"{val:g}"
    k = int(np.floor(np.log10(abs(err)))) # Stelle der ersten signifikanten
    ↪Ziffer
    r = round(err, -k) # Fehler auf eine signifikante
    ↪Ziffer runden
    fmt = f"{{:.{-k}f}} +/- {{:.{-k}f}}" if k < 0 else "{{:.0f}} +/- {{:.0f}}"
    return fmt.format(val, r)

# --- Parameter (anpassen) ---
lambda_nm = 635.0
sigma_lambda_nm = 0.0 # 0 falls vernachlaessigt
L = 1.500 # m
sigma_L = 0.001

# --- Messdaten (Beispiel): Abstand (-n) bis (+n) in mm ---
data_paare = pd.DataFrame({
    "n": [1, 2, 3, 4, 5, 6],
    "y_minus_mm": [-48.5, -96.0, -144.5, -192.0, -240.0, -289.5],
    "y_plus_mm": [ 48.0,  95.0,  144.0,  191.0,  240.5,  289.0]
})
```

```

df = data_paare.copy()
# Halber Abstand vom Zentrum: Mittelwert der Abweichungen links/rechts
df["y_half_mm"] = (df["y_plus_mm"] - df["y_minus_mm"]) / 2.0
D_mm = df["y_half_mm"].to_numpy()
n = df["n"].to_numpy().astype(float)
sigma_D_mm = np.full_like(D_mm, 1.0)

# --- Von D zu Winkeln ---
y = (D_mm * 1e-3) / 2.0
sigma_y = (sigma_D_mm * 1e-3) / 2.0
alpha = np.arctan2(y, L)
fac = 1.0 / (1.0 + (y / L) ** 2)
sigma_alpha = np.sqrt((fac * (1 / L) * sigma_y) ** 2 + (fac * (-y / L ** 2) *
    ↪ sigma_L) ** 2)

# --- Gewichteter linearer Fit: alpha = s*n + c (kein polyfit) ---
x, yv, w = n, alpha, 1.0 / (sigma_alpha ** 2)
enforce_through_origin = True # True: Fit durch den Ursprung erzwingen

if enforce_through_origin:
    Sxx = np.sum(w * x * x)
    Sxy = np.sum(w * x * yv)
    slope = Sxy / Sxx
    intercept = 0.0
    yhat = slope * x
    res = yv - yhat
    dof = len(x) - 1
    sigma2_hat = np.sum(w * res ** 2) / dof
    cov_s = sigma2_hat / Sxx
    slope_stderr = np.sqrt(cov_s)
    intercept_stderr = np.nan
else:
    Sw, Sx, Sy = np.sum(w), np.sum(w * x), np.sum(w * yv)
    Sxx, Sxy = np.sum(w * x * x), np.sum(w * x * yv)
    Delta = Sw * Sxx - Sx ** 2
    slope = (Sw * Sxy - Sx * Sy) / Delta
    intercept = (Sxx * Sy - Sx * Sxy) / Delta
    yhat = slope * x + intercept
    res = yv - yhat
    dof = len(x) - 2
    sigma2_hat = np.sum(w * res ** 2) / dof
    cov = sigma2_hat * np.array([[Sxx, -Sx], [-Sx, Sw]]) / Delta
    intercept_stderr = np.sqrt(cov[0, 0])
    slope_stderr = np.sqrt(cov[1, 1])

# --- Gewichtetes Bestimmtheitsmass R^2 ---

```

```

ybar_w = np.sum(w * yv) / np.sum(w)
SS_res_w = np.sum(w * (res ** 2))
SS_tot_w = np.sum(w * ((yv - ybar_w) ** 2))
R2 = 1.0 - SS_res_w / SS_tot_w

# --- Berechnung der Spaltbreite b ---
b = lambda_nm * 1e-9 / slope
db = np.sqrt((sigma_lambda_nm * 1e-9 / slope) ** 2 +
              (-(lambda_nm * 1e-9 / slope ** 2) * slope_stderr) ** 2)

# --- Ausgabe der Ergebnisse ---
print("Linearer Fit: alpha = s*n + c (gewichtete Ausgleichsgerade)")
print(f" s = {pm(slope, slope_stderr)} rad/Ordnung")
if not enforce_through_origin:
    print(f" c = {pm(intercept, intercept_stderr)} rad")
print(f" R^2 = {R2:.4f}")
print("\nSpaltbreite:")
print(f" b = {pm(b * 1e6, db * 1e6)} um")

# --- Darstellung ---
plt.figure()
plt.errorbar(x, yv * 1000, yerr=1000 / np.sqrt(w), fmt='o', capsize=3,
             label="Messung alpha_min,n")
xx = np.linspace(0, 1.05 * np.max(x), 200)
yy = slope * xx + (0 if enforce_through_origin else intercept)
plt.plot(xx, yy * 1000, label="Linearer Fit")
plt.xlabel("Ordnung n")
plt.ylabel("Winkel alpha_min,n [mrad]")
plt.title("Einzelspalt - Bestimmung der Spaltbreite b")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()

```

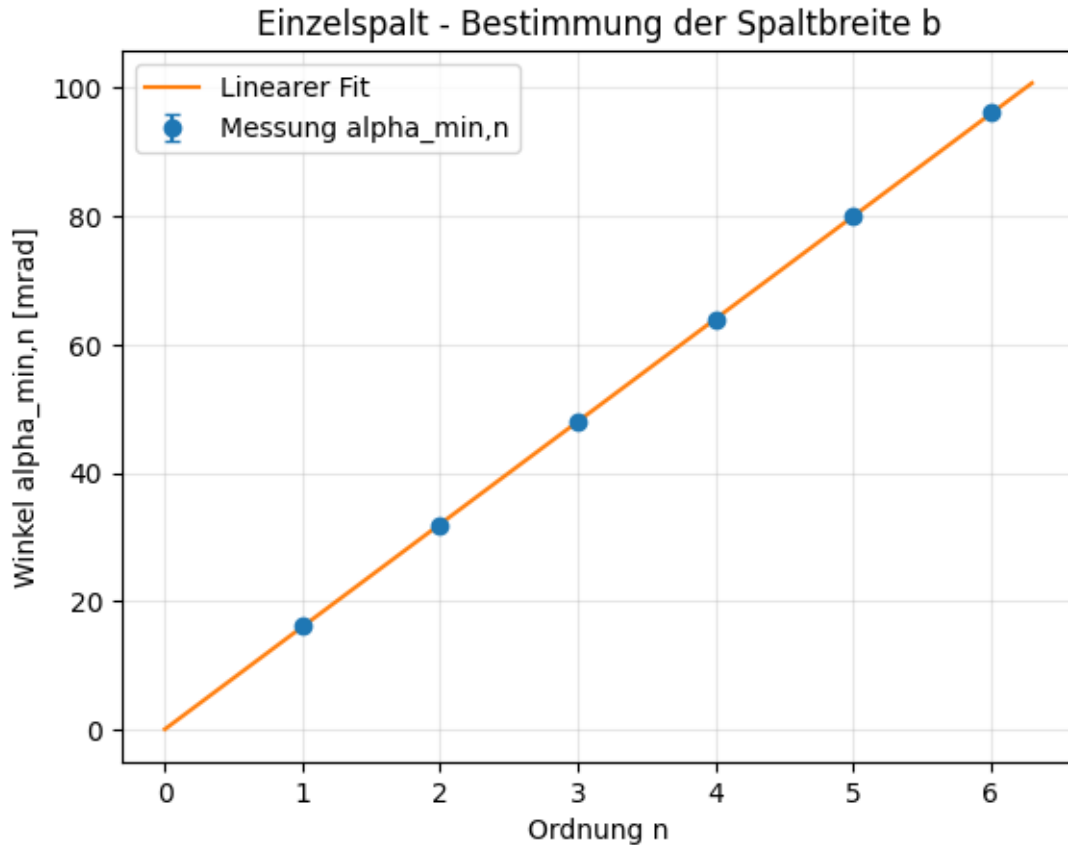
Linearer Fit: alpha = s*n + c (gewichtete Ausgleichsgerade)

s = 0.01599 +/- 0.00002 rad/Ordnung

R^2 = 1.0000

Spaltbreite:

b = 39.71 +/- 0.04 um



1.2 Experiment 2 – Beugung am Doppelspalt

1.2.1 Grundlagen

Beim **Doppelspalt** interferieren zwei kohärente Teilstrahlen, die durch die Spalte mit Abstand g treten.

Die Bedingung für **Interferenzmaxima** lautet:

$$g \sin(\alpha_{\max,k}) = k\lambda, \quad k = 0, 1, 2, \dots$$

und für **Minima**:

$$g \sin(\alpha_{\min,k}) = \frac{2k+1}{2} \lambda.$$

Bei realistischen Spalten überlagert sich dieses Interferenzmuster mit der Einzelspalt-Hüllkurve der Breite b :

$$I(\alpha) = I_0 \left(\frac{\sin(\beta)}{\beta} \right)^2 \cos^2(\gamma), \quad \beta = \frac{\pi b \sin(\alpha)}{\lambda}, \quad \gamma = \frac{\pi g \sin(\alpha)}{\lambda}.$$

1.2.2 Durchführung

Der Diodenlaser ($\lambda = 635 \text{ nm}$) beleuchtet den Doppelspalt senkrecht.

Das entstehende Beugungs- und Interferenzbild wird auf einem Schirm aufgefangen und die Positionen der Maxima bzw. Minima gemessen.

Zunächst wird analog zum Einzelspalt aus der Einhüllenden die Spaltbreite b bestimmt.

Anschließend wird der **Spaltabstand** g berechnet, indem gezählt wird, wie viele Interferenzmaxima („zweite Klasse“) innerhalb des zentralen Einzelspaltmaximums („erste Klasse“) auftreten.

Aus der Relation

$$\frac{b}{g} \approx \frac{k}{n}$$

folgt $g = b n / k$, wobei n die Ordnung des Minimums I. Klasse und k die Ordnung des Maximums II. Klasse bezeichnet.

1.2.3 Auswertung

1. Lineare Regression der Einzelspalt-Minima liefert b .
2. Aus der Zahl der Maxima II. Klasse im Hauptmaximum I. Klasse folgt g .
3. Die theoretische Intensitätsverteilung $I(\alpha)$ wird mit den Messwerten verglichen.

Die gemessenen Parameter b und g werden mit den Herstellerwerten verglichen und der relative Fehler angegeben.

```
[28]: # --- Importe ---
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# --- Hilfsfunktion fuer die Ausgabe im Format "x +/- dx" ---
def pm(val, err):
    if err == 0 or not np.isfinite(err):
        return f"{val:g}"
    k = int(np.floor(np.log10(abs(err))))
    r = round(err, -k)
    fmt = f"{{:.{-k}f}} +/- {{:.{-k}f}}" if k < 0 else "{:.0f} +/- {:.0f}"
    return fmt.format(val, r)

# --- Parameter ---
lambda_nm = 635.0      # nm
sigma_lambda_nm = 0.0
L = 1.500              # m
sigma_L = 0.001        # m

# =====
```

```

# TEIL 1: Doppelspalt - Bestimmung der Spaltbreite b
# =====
# Minima 1. Ordnung: aus der Einzelspalt-Huellkurve im Doppelspaltmuster
data_minima = pd.DataFrame({
    "n": [1, 2, 3, 4, 5],
    "y_minus_mm": [-11.8, -23.3, -35.0, -46.8, -58.6],
    "y_plus_mm": [ 11.8,  23.3,  34.8,  46.6,  58.4]
})
sigma_y_mm = 0.5
df_b = data_minima.copy()
df_b["y_half_mm"] = (df_b["y_plus_mm"] - df_b["y_minus_mm"]) / 2.0
df_b["sigma_y_half_mm"] = sigma_y_mm

n = df_b["n"].to_numpy(dtype=float)
y_b_m = df_b["y_half_mm"].to_numpy(dtype=float) * 1e-3
sigma_y_b_m = df_b["sigma_y_half_mm"].to_numpy(dtype=float) * 1e-3

# --- Umrechnung in den Winkel alpha ---
alpha_b = np.arctan2(y_b_m, L)
fac_b = 1.0 / (1.0 + (y_b_m / L) ** 2)
sigma_alpha_b = np.sqrt((fac_b * (1 / L) * sigma_y_b_m) ** 2 +
                        (fac_b * (-y_b_m / L ** 2) * sigma_L) ** 2)

# --- Lineare Regression: alpha = s * n ---
x, yv, w = n, alpha_b, 1.0 / (sigma_alpha_b ** 2)
Sxx = np.sum(w * x * x)
Sxy = np.sum(w * x * yv)
slope_b = Sxy / Sxx
res = yv - slope_b * x
dof = len(x) - 1
sigma2_hat = np.sum(w * res ** 2) / dof
slope_b_err = np.sqrt(sigma2_hat / Sxx)

# --- Bestimmung des Bestimmtheitsmasses R^2 ---
ybar_w = np.sum(w * yv) / np.sum(w)
R2_b = 1.0 - np.sum(w * (res ** 2)) / np.sum(w * ((yv - ybar_w) ** 2))

# --- Berechnung der Spaltbreite b ---
b = lambda_nm * 1e-9 / slope_b
db = np.sqrt((sigma_lambda_nm * 1e-9 / slope_b) ** 2 +
            ((-lambda_nm * 1e-9 / slope_b ** 2) * slope_b_err) ** 2)

# --- Ausgabe der Ergebnisse ---
print("=== Doppelspalt: Bestimmung der Spaltbreite b ===")
print(f"Steigung s = {pm(slope_b, slope_b_err)} rad/Ordnung")
print(f"R^2 = {R2_b:.4f}")
print(f"Spaltbreite b = {pm(b * 1e6, db * 1e6)} um\n")

```

```

# --- Darstellung der Messwerte und des Fits ---
plt.figure(figsize=(7, 5))
plt.errorbar(x, yv * 1e3, yerr=sigma_alpha_b * 1e3, fmt='o', capsize=3,
             label="Messung alpha_min,n")
xx = np.linspace(0, np.max(x) * 1.1, 200)
plt.plot(xx, slope_b * xx * 1e3, label="Linearer Fit")
plt.xlabel("Ordnung n")
plt.ylabel("Winkel alpha_min,n [mrad]")
plt.title("Doppelspalt - Bestimmung der Spaltbreite b")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()

# =====
# TEIL 2: Doppelspalt - Bestimmung des Spaltabstands g
# =====
# Maxima 2. Ordnung: symmetrische k-te Maxima links und rechts
data_maxima = pd.DataFrame({
    "k": [1, 2, 3, 4, 5],
    "y_minus_mm": [-20.5, -41.0, -61.5, -82.0, -102.5],
    "y_plus_mm": [ 20.5,  41.0,  61.0,  82.0, 102.0]
})

df_g = data_maxima.copy()
df_g["y_half_mm"] = (df_g["y_plus_mm"] - df_g["y_minus_mm"]) / 2.0
sigma_y_mm = 0.5
df_g["sigma_y_half_mm"] = sigma_y_mm

k = df_g["k"].to_numpy(dtype=float)
y_half_m = df_g["y_half_mm"].to_numpy(dtype=float) * 1e-3
sigma_y_half_m = df_g["sigma_y_half_mm"].to_numpy(dtype=float) * 1e-3

# --- Umrechnung in den Winkel alpha ---
alpha = np.arctan2(y_half_m, L)
fac = 1.0 / (1.0 + (y_half_m / L) ** 2)
sigma_alpha = np.sqrt((fac * (1 / L) * sigma_y_half_m) ** 2 +
                      (fac * (-y_half_m / L ** 2) * sigma_L) ** 2)

# --- Lineare Regression: alpha = s * k ---
x, yv, w = k, alpha, 1.0 / (sigma_alpha ** 2)
Sxx = np.sum(w * x * x)
Sxy = np.sum(w * x * yv)
slope_g = Sxy / Sxx
res = yv - slope_g * x
dof = len(x) - 1
sigma2_hat = np.sum(w * res ** 2) / dof

```

```

slope_g_err = np.sqrt(sigma2_hat / Sxx)

# --- Bestimmung des Bestimmtheitsmasses  $R^2$  ---
ybar_w = np.sum(w * yv) / np.sum(w)
R2_g = 1.0 - np.sum(w * (res ** 2)) / np.sum(w * ((yv - ybar_w) ** 2))

# --- Berechnung des Spaltabstands  $g$  ---
g = lambda_nm * 1e-9 / slope_g
dg = np.sqrt((sigma_lambda_nm * 1e-9 / slope_g) ** 2 +
              ((-lambda_nm * 1e-9 / slope_g ** 2) * slope_g_err) ** 2)

# --- Ausgabe der Ergebnisse ---
print("=== Doppelspalt: Bestimmung des Spaltabstands  $g$  ===")
print(f"Steigung  $s = \{pm(slope\_g, slope\_g\_err)\}$  rad/Ordnung")
print(f" $R^2 = \{R2\_g:.4f\}$ ")
print(f"Spaltabstand  $g = \{pm(g * 1e6, dg * 1e6)\}$  um\n")

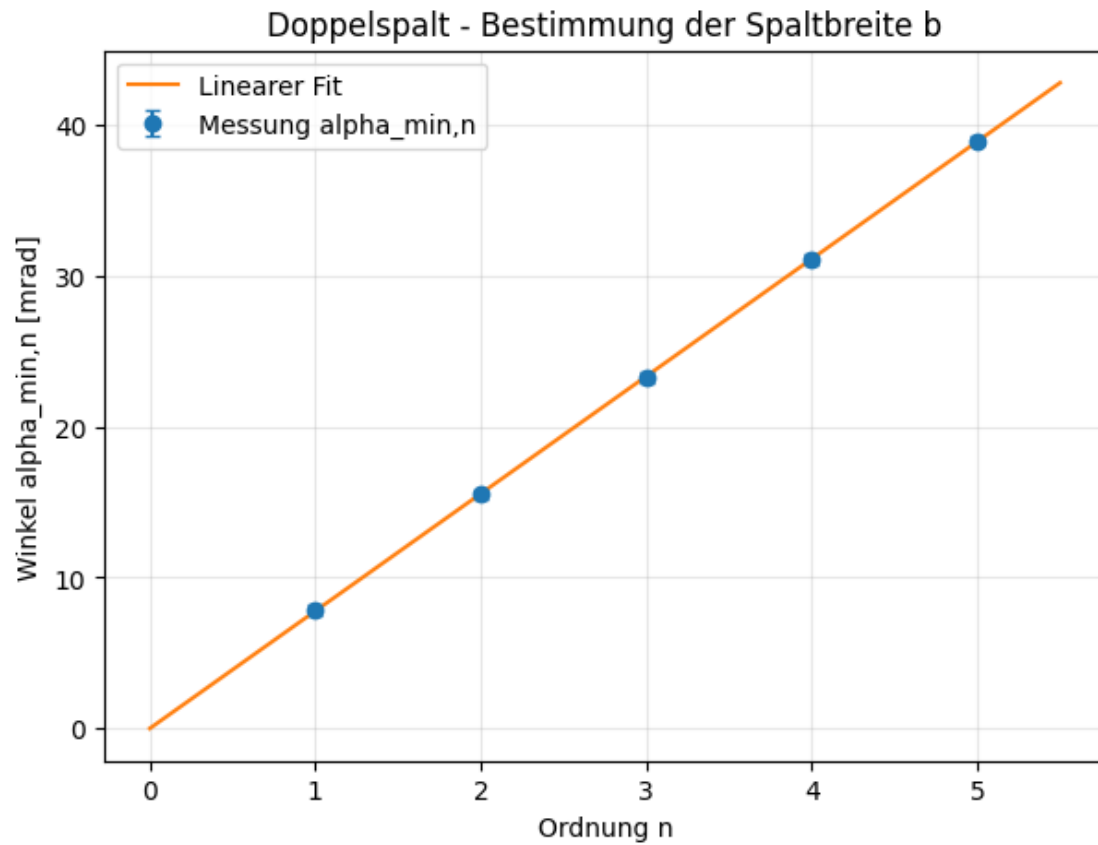
# --- Darstellung der Messwerte und des Fits ---
plt.figure(figsize=(7, 5))
plt.errorbar(x, yv * 1e3, yerr=sigma_alpha * 1e3, fmt='o', capsize=3,
             label="Messung alpha_max,k")
xx = np.linspace(0, np.max(x) * 1.1, 200)
plt.plot(xx, slope_g * xx * 1e3, label="Linearer Fit")
plt.xlabel("Ordnung k")
plt.ylabel("Winkel alpha_max,k [mrad]")
plt.title("Doppelspalt - Bestimmung des Spaltabstands  $g$ ")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()

```

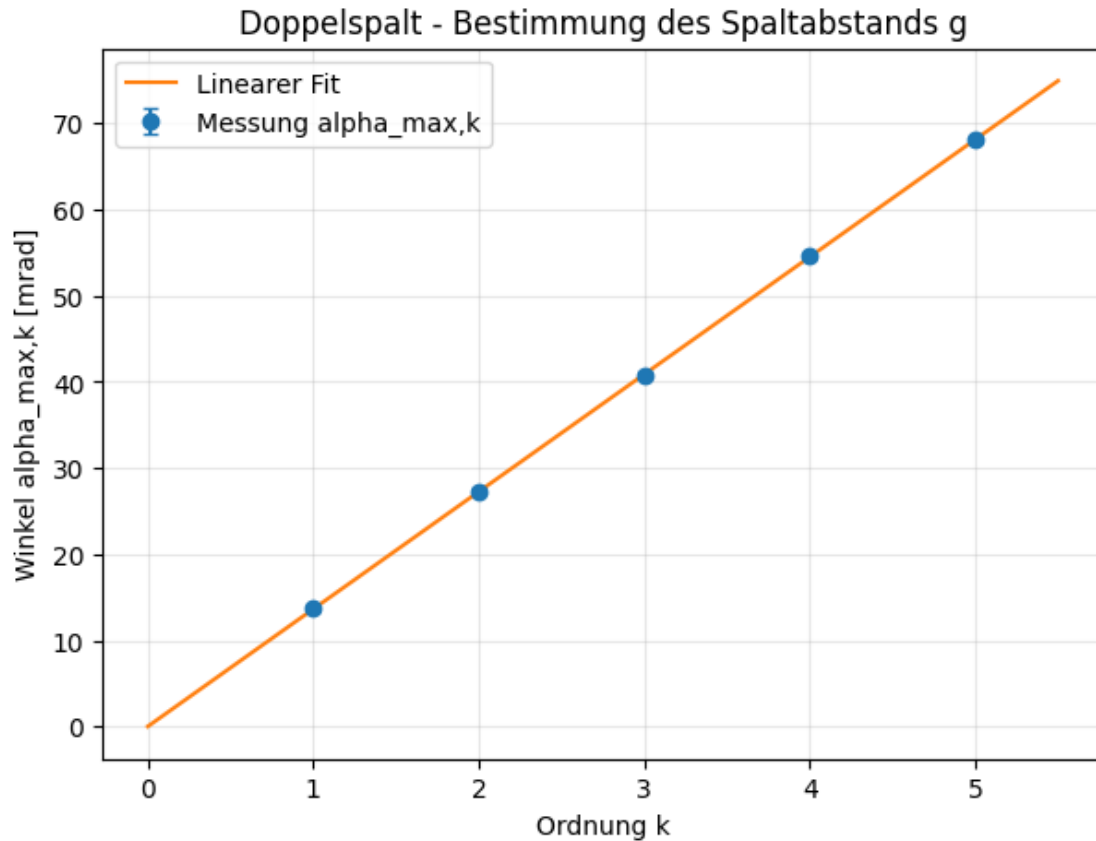
```

=== Doppelspalt: Bestimmung der Spaltbreite  $b$  ===
Steigung  $s = 0.007784 \pm 0.000010$  rad/Ordnung
 $R^2 = 1.0000$ 
Spaltbreite  $b = 81.58 \pm 0.10$  um

```



=== Doppelspalt: Bestimmung des Spaltabstands g ===
Steigung $s = 0.01363 \pm 0.00001$ rad/Ordnung
 $R^2 = 1.0000$
Spaltabstand $g = 46.60 \pm 0.04$ μm



1.3 Experiment 3 – Wellenlängenmessung mit dem Beugungsgitter

1.3.1 Grundlagen

Ein **Beugungsgitter** besteht aus vielen äquidistanten Spalten mit der **Gitterkonstanten** g . Bei Beugung monochromatischen Lichts entsteht Interferenz, wenn der Gangunterschied zwischen benachbarten Spalten ein ganzzahliges Vielfaches der Wellenlänge ist:

$$g \sin(\alpha_k) = k\lambda, \quad k = 0, 1, 2, \dots$$

Diese Beziehung ist die **Gittergleichung**.

Für gegebene Ordnung k kann daraus entweder g oder λ bestimmt werden.

1.3.2 Durchführung

Das Gitter wird zunächst mit einem **Laser** bekannter Wellenlänge (z. B. $\lambda = 635 \text{ nm}$) beleuchtet, um die Gitterkonstante g experimentell zu bestimmen.

Dazu wird der Ablenkwinkel der 1. Ordnung gemessen:

$$g = \frac{\lambda}{\sin(\alpha_1)}.$$

Anschließend wird eine **Spektrallampe** verwendet.

Das entstehende Spektrum wird mit einem **Präzisionsgoniometer** beobachtet.

Die Winkel der Spektrallinien für mehrere Ordnungen $k = 1, 2, 3$ werden beidseitig des Zentralmaximums gemessen.

Der mittlere Beugungswinkel wird als:

$$\alpha_k = \frac{1}{2}(\alpha_{\text{rechts}} - \alpha_{\text{links}})$$

bestimmt.

1.3.3 Auswertung

Aus der Gittergleichung folgt für jede Linie:

$$\lambda = g \sin(\alpha_k) / k.$$

Für jede Spektrallinie wird der Mittelwert der aus den Ordnungen erhaltenen Wellenlängen berechnet.

Die Ergebnisse werden mit den Referenzwerten der **Tabelle 1 (Spektrallinien bekannter Elemente)** verglichen, um das verwendete Element (z. B. Hg, Na, Cd) zu identifizieren.

Die relative Abweichung $\Delta\lambda/\lambda_{\text{tab}}$ wird in % angegeben.

```
[29]: # --- Wellenlaengemessung mit dem Gitter ---
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# --- Hilfsfunktion fuer die Ausgabe im Format "x +/- dx" ---
def pm(val, err):
    if err == 0 or not np.isfinite(err):
        return f"{val:g}"
    k = int(np.floor(np.log10(abs(err))))
    r = round(err, -k)
    fmt = f"{{:.{-k}f}} +/- {{:.{-k}f}}" if k < 0 else "{{:.0f}} +/- {{:.0f}}"
    return fmt.format(val, r)

# ===== 1. Bestimmung der Gitterkonstante g =====
# Bekannte Wellenlaenge des Lasers (z.B. He-Ne-Laser)
lambda_laser_nm = 635.0
sigma_lambda_laser_nm = 0.5
lambda_laser_m = lambda_laser_nm * 1e-9

# Messwerte fuer die Beugung des Lasers (k = 1)
alpha_right = 20
alpha_left = -20
alpha_deg = (alpha_right - alpha_left) / 2 # gemessener Beugungswinkel
sigma_alpha_deg = 0.2
```

```

k = 1

# Umrechnung von Grad in Radiant
alpha_rad = np.deg2rad(alpha_deg)
sigma_alpha_rad = np.deg2rad(sigma_alpha_deg)

# Berechnung der Gitterkonstante:  $g = k * \lambda / \sin(\alpha)$ 
g_m = k * lambda_laser_m / np.sin(alpha_rad)

# Fehlerfortpflanzung fuer g
sigma_g_m = g_m * np.sqrt(
    (sigma_lambda_laser_nm / lambda_laser_nm) ** 2 +
    (np.cos(alpha_rad) / np.sin(alpha_rad) * sigma_alpha_rad) ** 2
)

# Ausgabe der berechneten Gitterkonstante
print("=== Gitterkonstante aus Laser-Messung ===")
print(f"g = {pm(g_m * 1e6, sigma_g_m * 1e6)} um")

# ===== 2. Bestimmung der Wellenlaengen aus der Spektrallampe =====
# Beispielhafte Messdaten der Spektrallampe
data = pd.DataFrame({
    "name": ["gelb", "gelb", "gruen", "gruen", "blau"],
    "k": [1, 2, 1, 2, 1],
    "alpha_links": [-18.90, -39.10, -20.15, -41.95, -14.10],
    "alpha_rechts": [18.95, 39.05, 20.20, 42.05, 14.05],
    "sigma_alpha_deg": [0.2, 0.2, 0.2, 0.2, 0.2]
})

# Berechnung der mittleren Beugungswinkel aus den Messwerten
data["alpha_deg"] = (data["alpha_rechts"] - data["alpha_links"]) / 2
data["alpha_rad"] = np.deg2rad(data["alpha_deg"])
data["sigma_alpha_rad"] = np.deg2rad(data["sigma_alpha_deg"])

# Berechnung der Wellenlaengen:  $\lambda = g * \sin(\alpha) / k$ 
data["lambda_m"] = g_m * np.sin(data["alpha_rad"]) / data["k"]
data["sigma_lambda_m"] = np.sqrt(
    (np.sin(data["alpha_rad"]) / data["k"] * sigma_g_m) ** 2 +
    (g_m * np.cos(data["alpha_rad"]) / data["k"] * data["sigma_alpha_rad"]) ** 2
)

# Umrechnung der Wellenlaengen in Nanometer
data["lambda_nm"] = data["lambda_m"] * 1e9
data["sigma_lambda_nm"] = data["sigma_lambda_m"] * 1e9

# Ausgabe der berechneten Wellenlaengen
print("\n=== Wellenlaengen der Spektrallinien ===")

```

```

for _, row in data.iterrows():
    print(f"{row['name']:>5s}: lambda = {pm(row['lambda_nm'],
    ↳row['sigma_lambda_nm'])} nm")

# ===== 3. Plot der Ergebnisse =====
# Die folgenden Zeilen koennen aktiviert werden, um die Messwerte grafisch
↳darzustellen.
# plt.figure(figsize=(7, 4))
# plt.errorbar(
#     data["alpha_deg"], data["lambda_nm"],
#     yerr=data["sigma_lambda_nm"], xerr=data["sigma_alpha_deg"],
#     fmt="o", capsize=4, label="Spektrallinien"
# )
# plt.title("Wellenlaengen aus Beugungswinkeln (Gitter bestimmt aus Laser)")
# plt.xlabel("Beugungswinkel alpha (Grad)")
# plt.ylabel("Wellenlaenge lambda (nm)")
# plt.grid(True)
# plt.legend()
# plt.show()

```

=== Gitterkonstante aus Laser-Messung ===

$g = 1.86 \pm 0.02 \text{ um}$

=== Wellenlaengen der Spektrallinien ===

gelb: $\lambda = 602 \pm 8 \text{ nm}$
 gelb: $\lambda = 585 \pm 6 \text{ nm}$
 gruen: $\lambda = 640 \pm 9 \text{ nm}$
 gruen: $\lambda = 621 \pm 6 \text{ nm}$
 blau: $\lambda = 452 \pm 8 \text{ nm}$

Element	λ/nm	Farbeindruck	Helligkeit
Natrium (Na)	616.08	gelbrot	mittel
	615.42	gelbrot	mittel
	589.59	gelb	stark
	589.00	gelb	mittel
	568.82	gelbgrün	mittel
	568.27	gelbgrün	mittel
Kalium (K)	769.90	dunkelrot	stark
	766.40	dunkelrot	stark
	404.72	violett	mittel
	404.41	violett	mittel
Cadmium (Cd)	643.85	rot	stark
	635.99	gelbrot	schwach
	508.58	grün	stark
	479.99	blaugrün	stark
	467.82	blau	stark
	441.46	blau	mittel
Quecksilber (Hg)	708.19	rot	schwach
	690.72	gelbrot	schwach
	579.07	grün	sehr stark
	576.96	blaugrün	sehr stark
	546.07	blau	stark
	491.60	blau	mittel
	435.84	blaugrün	stark
	407.78	blau	mittel
	404.66	blau	mittel

Unser Meinung nach, muss es <> sein