lab2

October 14, 2025

1 PW2 – Geometrische Optik

1.1 Experiment 1: Brennweite von Linsen

1.1.1 Grundlagen

Die **geometrische Optik (Strahlenoptik)** beschreibt die Ausbreitung des Lichtes anhand geradliniger Strahlen. Wechselwirkungen mit Materie führen nur zu Richtungsänderungen durch **Reflexion** und **Brechung**.

Das **Reflexionsgesetz** besagt: > Der Einfallswinkel ist gleich dem Reflexionswinkel ,
> und einfallender Strahl, reflektierter Strahl und Lot liegen in einer Ebene.

Das **Brechungsgesetz** lautet:

$$\frac{\sin\alpha}{\sin\beta} = \frac{c_1}{c_2} = \frac{n_2}{n_1} = n_{21}$$

Ein Übergang vom optisch dünneren ins dichtere Medium führt zur Brechung **zum Lot hin**, umgekehrt **vom Lot weg**.

Linsen Eine **Linse** besteht aus mindestens zwei brechenden Grenzflächen, von denen mindestens eine gekrümmt ist.
- **Konvexlinsen (Sammellinsen):** sammeln parallele Strahlen im Brennpunkt F
- **Konkavlinsen (Zerstreuungslinsen):** zerstreuen Strahlen, als kämen sie von einem vor der Linse liegenden Brennpunkt

Die **Linsengleichung** lautet:

$$\frac{1}{f} = \frac{1}{g} + \frac{1}{b}$$

Der **Abbildungsmaßstab** ist:

$$V = \frac{B}{G} = \frac{b}{g}$$

Für dünne Linsen in Luft gilt:

$$\frac{1}{f} = (n-1)\left(\frac{1}{r_1} + \frac{1}{r_2}\right)$$

Besselverfahren Bei festem Abstand e zwischen Gegenstand und Schirm gibt es zwei symmetrische Linsenpositionen, in denen scharfe Bilder entstehen, wenn $ e 4f $.
Mit der Verschiebung d der Linsenpositionen gilt:

$$f = \frac{1}{4}\left(e - \frac{d^2}{e}\right)$$

1.1.2 Versuchsaufbau, Durchführung, und Auswertung

- Montiere auf einer **optischen Bank**:
  - Lichtquelle (6 V-Versorgung, **nicht direkt an Netzspannung!**)
  - Gegenstand (Mattglasscheibe mit Strichmarken)
  - optischer Schirm (Mattglasscheibe)
- Justiere alle Komponenten so, dass die **optische Achse** durch die Mitte der Linse(n) geht.

Aufgabe 1a: Direkte Brennweitenmessung

1. Stelle eine feste Gegenstandsweite g ein.

2. Erzeuge ein scharfes Bild und miss die Bildweite b.

3. Berechne f aus der Linsengleichung.

4. Wiederhole mindestens fünfmal, um Mittelwert und Unsicherheit zu bestimmen.

Aufgabe 1b: Besselverfahren

1. Wähle fünf verschiedene Abstände e zwischen Gegenstand und Schirm.

2. Miss die Verschiebung d der Linse.

3. Berechne f mit obiger Formel.

[11]:
```python
"""
Geometrische Optik - Teil 1 (1a und 1b)

1a: Brennweite über Linsengleichung
1b: Bessel-Methode
"""

import math
import numpy as np
from typing import List, Tuple

# =========================
# EINGABEDATEN (mm)
# =========================
# Teil 1a - mehrere Messreihen:
# x_G = Objektposition, x_L = Linse, x_S = Schirm (bei scharfem Bild)
x_G_1a = [1601.0, 1601.0, 1601.0, 1601.0, 1601.0, 1601.0]
```

```python
x_L_1a = [1300, 1300, 1300, 1300, 1300, 1300]
x_S_1a = [995.5, 999.0, 999.0, 998.5, 998.0, 1109.0]

# Teil 1b (Bessel) - mehrere Serien:
# x_G = Objekt, x_S = Schirm (fix), x_L1 / x_L2 = zwei scharfe Linsenpositionen
x_G_1b = [1601.0, 1601.0, 1601.0, 1601.0, 1601.0]
x_S_1b = [1000.0, 950.0, 900.0, 850.0, 800.0]
x_L1_1b = [1300.0, 1370.0, 1386.0, 1394.0, 1404.0]
x_L2_1b = [1270.0, 1188.0, 1129.0, 1063.0, 1005.0]

# =========================
# UNSICHERHEITEN DER KOORDINATEN (mm)
# =========================
SIGMA_LESEN = 0.5    # Ablesefehler/Parallaxe
SIGMA_FOKUS = 1.0    # Subjektives Scharfstellen

# =========================
# HILFSFUNKTIONEN
# =========================
def diff_sigma(s1: float, s2: float) -> float:
    return math.hypot(s1, s2)

def f_aus_gb(g: float, b: float) -> float:
    return (g*b)/(g+b)

def sigma_f_aus_gb(g: float, b: float, sg: float, sb: float) -> float:
    denom2 = (g+b)**2
    dfdg = (b*b)/denom2
    dfdb = (g*g)/denom2
    return math.hypot(dfdg*sg, dfdb*sb)

def f_bessel(e: float, d: float) -> float:
    return 0.25*(e - (d*d)/e)

def sigma_f_bessel(e: float, d: float, se: float, sd: float) -> float:
    dfdE = 0.25*(1.0 + (d*d)/(e*e))
    dfdD = -0.5*(d/e)
    return math.hypot(dfdE*se, dfdD*sd)

def kombi_sigma(stat: float, instr: float, mode: str="max") -> float:
    return max(stat, instr) if mode == "max" else math.hypot(stat, instr)

def dioptrien(f_mm: float, sf_mm: float) -> Tuple[float, float]:
    f_m = f_mm/1000.0
    D = 1.0/f_m
    sD = (sf_mm/1000.0)/(f_m*f_m)
    return D, sD
```

```python
def     fmt_pm(val: float, unc: float, einheit: str) -> str:
    if unc <= 0 or not math.isfinite(unc):
        return f"{val:.3g} {einheit}"
    e = int(math.floor(math.log10(abs(unc)))) if unc != 0 else 0
    lead = round(unc/(10**e))
    n_sig = 2 if lead == 1 else 1
    prec = -e + (n_sig - 1)
    val_r = round(val, prec)
    unc_r = round(unc, prec)
    return f"{val_r:.{max(0,prec)}f} ± {unc_r:.{max(0,prec)}f} {einheit}"

# ==========================
# TEIL 1a - LINSENGLEICHUNG
# ==========================
print("\n=== TEIL 1a - LINSENGLEICHUNG ===")
N1 = len(x_G_1a)
assert len(x_L_1a)==N1 and len(x_S_1a)==N1, "Längen der Arrays 1a stimmen nicht.
 ↪"

# Koordinaten-Unsicherheiten
sigma_xG = SIGMA_LESEN
sigma_xL = math.hypot(SIGMA_LESEN, 0.5*SIGMA_FOKUS)
sigma_xS = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)
sigma_g = diff_sigma(sigma_xL, sigma_xG)
sigma_b = diff_sigma(sigma_xS, sigma_xL)

f_werte, f_instr = [], []
for i in range(N1):
    g = abs(x_L_1a[i] - x_G_1a[i])
    b = abs(x_S_1a[i] - x_L_1a[i])
    f_i = f_aus_gb(g, b)
    sf_i = sigma_f_aus_gb(g, b, sigma_g, sigma_b)
    f_werte.append(f_i)
    f_instr.append(sf_i)
    print(f"[Messung {i+1}] g={g:.2f} mm, b={b:.2f} mm -> f={f_i:.3f} mm,␣
 ↪ _f(instr)={sf_i:.3f} mm")

f_mittel = float(np.mean(f_werte))
f_stat = float(np.std(f_werte, ddof=1)) if N1>=2 else float(np.mean(f_instr))
f_instr_mittel = float(np.mean(f_instr))
f_sigma = kombi_sigma(f_stat, f_instr_mittel)
D, sD = dioptrien(f_mittel, f_sigma)

print("\nERGEBNIS 1a:")
print(f"   (x_G)={sigma_xG:.3g},  (x_L)={sigma_xL:.3g},  (x_S)={sigma_xS:.3g}")
print(f"   (g)={sigma_g:.3g},  (b)={sigma_b:.3g}")
```

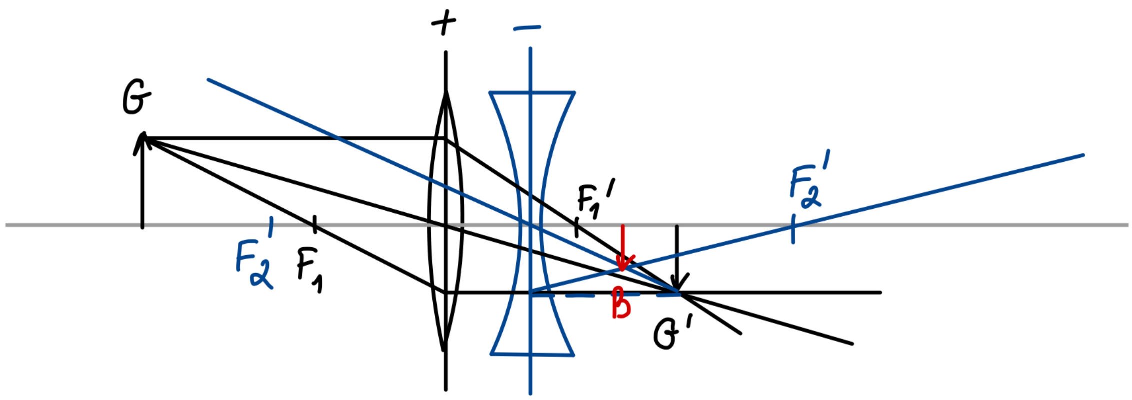
```python
print("  f =", fmt_pm(f_mittel, f_sigma, "mm"))
print("  D  =", fmt_pm(D, sD, "dpt"))
print(f"  (Stat.)   = {f_stat:.3g} mm; (Instr.)   = {f_instr_mittel:.3g} mm")
print("  Hinweis: Endunsicherheit = max(statistisch, instrumentell).")

# ========================
# TEIL 1b - BESSEL-METHODE
# ========================
print("\n=== TEIL 1b - BESSEL-METHODE ===")
N2 = len(x_G_1b)
assert len(x_S_1b)==N2 and len(x_L1_1b)==N2 and len(x_L2_1b)==N2, "Längen der Arrays 1b stimmen nicht."

sigma_xG_b = SIGMA_LESEN
sigma_xS_b = SIGMA_LESEN
sigma_xL1_b = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)
sigma_xL2_b = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)
sigma_e = diff_sigma(sigma_xS_b, sigma_xG_b)
sigma_d = diff_sigma(sigma_xL2_b, sigma_xL1_b)

f_werte_b, f_instr_b = [], []
for i in range(N2):
    e = abs(x_S_1b[i] - x_G_1b[i])
    d = abs(x_L2_1b[i] - x_L1_1b[i])
    f_i = f_bessel(e, d)
    sf_i = sigma_f_bessel(e, d, sigma_e, sigma_d)
    f_werte_b.append(f_i)
    f_instr_b.append(sf_i)
    print(f"[Serie {i+1}] e={e:.2f} mm, d={d:.2f} mm -> f={f_i:.3f} mm,  _f(instr)={sf_i:.3f} mm")

f_mittel_b = float(np.mean(f_werte_b))
f_stat_b = float(np.std(f_werte_b, ddof=1)) if N2>=2 else float(np.mean(f_instr_b))
f_instr_mittel_b = float(np.mean(f_instr_b))
f_sigma_b = kombi_sigma(f_stat_b, f_instr_mittel_b)
D_b, sD_b = dioptrien(f_mittel_b, f_sigma_b)

print("\nERGEBNIS 1b:")
print(f"   (x_G)={sigma_xG_b:.3g},  (x_S)={sigma_xS_b:.3g},  (x_L1)={sigma_xL1_b:.3g},  (x_L2)={sigma_xL2_b:.3g}")
print(f"   (e)={sigma_e:.3g},  (d)={sigma_d:.3g}")
print("  f =", fmt_pm(f_mittel_b, f_sigma_b, "mm"))
print("  D  =", fmt_pm(D_b, sD_b, "dpt"))
print(f"  (Stat.)   = {f_stat_b:.3g} mm; (Instr.)   = {f_instr_mittel_b:.3g} mm")
print("  Hinweis: Bei großen e ist  _instr der Bessel-Methode meist sehr klein.")
```

```
=== TEIL 1a - LINSENGLEICHUNG ===
[Messung 1] g=301.00 mm, b=304.50 mm -> f=151.370 mm,  _f(instr)=0.393 mm
[Messung 2] g=301.00 mm, b=301.00 mm -> f=150.500 mm,  _f(instr)=0.395 mm
[Messung 3] g=301.00 mm, b=301.00 mm -> f=150.500 mm,  _f(instr)=0.395 mm
[Messung 4] g=301.00 mm, b=301.50 mm -> f=150.625 mm,  _f(instr)=0.395 mm
[Messung 5] g=301.00 mm, b=302.00 mm -> f=150.750 mm,  _f(instr)=0.395 mm
[Messung 6] g=301.00 mm, b=191.00 mm -> f=116.852 mm,  _f(instr)=0.512 mm

ERGEBNIS 1a:
   (x_G)=0.5,  (x_L)=0.707,  (x_S)=1.12
   (g)=0.866,  (b)=1.32
  f = 145 ± 14 mm
  D  = 6.9 ± 0.7 dpt
  (Stat.)   = 13.8 mm; (Instr.)   = 0.414 mm
  Hinweis: Endunsicherheit = max(statistisch, instrumentell).

=== TEIL 1b - BESSEL-METHODE ===
[Serie 1] e=601.00 mm, d=30.00 mm -> f=149.876 mm,  _f(instr)=0.182 mm
[Serie 2] e=651.00 mm, d=182.00 mm -> f=150.030 mm,  _f(instr)=0.292 mm
[Serie 3] e=701.00 mm, d=257.00 mm -> f=151.695 mm,  _f(instr)=0.352 mm
[Serie 4] e=751.00 mm, d=331.00 mm -> f=151.278 mm,  _f(instr)=0.407 mm
[Serie 5] e=801.00 mm, d=399.00 mm -> f=150.562 mm,  _f(instr)=0.451 mm

ERGEBNIS 1b:
   (x_G)=0.5,  (x_S)=0.5,  (x_L1)=1.12,  (x_L2)=1.12
   (e)=0.707,  (d)=1.58
  f = 150.7 ± 0.8 mm
  D  = 6.64 ± 0.03 dpt
  (Stat.)   = 0.786 mm; (Instr.)   = 0.337 mm
  Hinweis: Bei großen e ist  _instr der Bessel-Methode meist sehr klein.
```

Aufgabe 2: Brennweite einer Konkavlinse

- Kombiniere eine **Konvexlinse** (liefert reelles Zwischenbild) mit der **Konkavlinse**.
- Bestimme:
  - g_1, b_1 für die Konvexlinse
  - $g_2 = -(b_1 - d)$, b_2 für die Konkavlinse

  - Berechne f aus der Linsengleichung.
- Erstelle eine Skizze des Strahlenganges.



[12]:
```
# ==================================================================
# TEIL 2 - ZERSTREUUNGSLINSE (mit Sammellinse als Vorstufe)
# Schema eines Durchgangs:
#   1) Nur Sammellinse: scharfes (verkleinertes) Bild -> messe x_G, x_L_plus,␣
 ↪x_S1
#      g1 = |x_L+ - x_G|, b1 = |x_S1 - x_L+|
#   2) Zerstreuungslinse hinzufügen: Abstand d zwischen Linsen; Schirm -> neue␣
 ↪Schärfe bei x_S2
#      g2 = -(b1 - d); b2 = |x_S2 - x_L-|
#      f- (negativ) aus 1/f = 1/g2 + 1/b2
# Wir erlauben mehrere Durchgänge parallel (Arrays).
# ==================================================================

print("\n=== TEIL 2 - ZERSTREUUNGSLINSE ===")

# -------------------------
# EINGABEDATEN TEIL 2 (mm)
# -------------------------
# Für jeden Durchgang i folgende Arrays gleicher Länge:
# Objektposition:
x_G_t2    = [1601.0]
# Sammellinse-Position (L+):
x_Lplus_t2 = [1230.5]
# Schirmposition mit NUR L+ (scharf):
x_S1_t2   = [980.0]
# Zerstreuungslinse-Position (L-):
x_Lminus_t2 = [1180.5]
# Abstand Linsen (optional direkter Wert; falls None, wird aus Koordinaten␣
 ↪berechnet)
# d = |x_Lminus - x_Lplus|
d_override = [None]
# Schirmposition mit L+ und L- (neue Schärfe):
x_S2_t2   = [832.0]
```

```
N2 = len(x_G_t2)
assert all(len(arr)==N2 for arr in [x_Lplus_t2, x_S1_t2, x_Lminus_t2, x_S2_t2,␣
 ↪d_override]), "Array-Längen in TEIL 2 ungleich."

# Unsicherheiten einzelner Koordinaten:
sigma_xG = SIGMA_LESEN
sigma_xLplus = math.hypot(SIGMA_LESEN, 0.5*SIGMA_FOKUS)  # Sammellinse gering␣
 ↪fokusbeeinflusst
sigma_xS1  = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)         # Schirm bei Schärfe
sigma_xLminus = math.hypot(SIGMA_LESEN, 0.5*SIGMA_FOKUS) # Zerstreuungslinse␣
 ↪(Position fest)
sigma_xS2 = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)

# Abgeleitete Unsicherheiten:
sigma_g1 = diff_sigma(sigma_xLplus, sigma_xG)
sigma_b1 = diff_sigma(sigma_xS1,   sigma_xLplus)
sigma_d  = diff_sigma(sigma_xLminus, sigma_xLplus)
sigma_b2 = diff_sigma(sigma_xS2,   sigma_xLminus)

f_minus_vals = []
sigma_f_minus_instr = []

for i in range(N2):
    g1 = abs(x_Lplus_t2[i] - x_G_t2[i])
    b1 = abs(x_S1_t2[i]    - x_Lplus_t2[i])
    d  = abs(x_Lminus_t2[i] - x_Lplus_t2[i]) if d_override[i] is None else␣
 ↪float(d_override[i])
    g2 = -(b1 - d)  # virtuelles Objekt für L-
    b2 = abs(x_S2_t2[i] - x_Lminus_t2[i])

    # Unsicherheit von g2 aus b1 und d:
    sigma_g2 = diff_sigma(sigma_b1, sigma_d)

    # Zerstreuungslinse: f- aus 1/f = 1/g2 + 1/b2
    # -> f = (g2*b2)/(g2 + b2)
    f_minus = f_aus_gb(g2, b2)  # g2 < 0 sollte negatives f ergeben
    # Instrumentelle Unsicherheit über  f/ g2,  f/ b2
    sigma_fm = sigma_f_aus_gb(g2, b2, sigma_g2, sigma_b2)

    f_minus_vals.append(f_minus)
    sigma_f_minus_instr.append(sigma_fm)

    print(f"[Durchgang {i+1}] g1={g1:.2f} mm, b1={b1:.2f} mm, d={d:.2f} mm ->␣
 ↪g2={g2:.2f} mm, b2={b2:.2f} mm")
    print(f"    f- = {f_minus:.3f} mm (neg.),  _f(instr) = {sigma_fm:.3f} mm")

# Statistische Kenngrößen
```

```
f_minus_mean = float(np.mean(f_minus_vals))
f_minus_stat = float(np.std(f_minus_vals, ddof=1)) if N2>=2 else float(np.
  ↪mean(sigma_f_minus_instr))
f_minus_instr_mean = float(np.mean(sigma_f_minus_instr))

# Endunsicherheit als Max(stat, instr)
f_minus_sigma = max(f_minus_stat, f_minus_instr_mean)
D_minus, sD_minus = dioptrien(f_minus_mean, f_minus_sigma)

print("\nERGEBNIS TEIL 2:")
print(f"  (g1)={sigma_g1:.3g} mm,  (b1)={sigma_b1:.3g} mm,  (d)={sigma_d:.3g}␣
  ↪mm,  (b2)={sigma_b2:.3g} mm")
print("  f- =", fmt_pm(f_minus_mean, f_minus_sigma, "mm"))
print("  D-  =", fmt_pm(D_minus, sD_minus, "dpt"))
print(f"  (Stat.)   = {f_minus_stat:.3g} mm; (Instr.)   = {f_minus_instr_mean:.
  ↪3g} mm")
print("  Hinweis: f- sollte negativ sein (Zerstreuungslinse).")
```

```
=== TEIL 2 - ZERSTREUUNGSLINSE ===
[Durchgang 1] g1=370.50 mm, b1=250.50 mm, d=50.00 mm -> g2=-200.50 mm, b2=348.50
mm
    f- = -472.123 mm (neg.),  _f(instr) = 9.510 mm

ERGEBNIS TEIL 2:
   (g1)=0.866 mm,  (b1)=1.32 mm,  (d)=1 mm,  (b2)=1.32 mm
  f- = -472 ± 10 mm
  D-  = -2.12 ± 0.04 dpt
  (Stat.)   = 9.51 mm; (Instr.)   = 9.51 mm
  Hinweis: f- sollte negativ sein (Zerstreuungslinse).
```

Aufgabe 3: Brechkraft der untersuchten Linsen Berechne nach den Brennweitenmessungen die **Brechkraft** D jeder Linse in **Dioptrien**:

$$D = \frac{1}{f}$$

f in Metern, D in $[\mathrm{m}\ ^{1}]$ = Dioptrien

[13]:
```
"""
TEIL 3 - DIOPTRISCHE STÄRKE (D = 1/f) - im selben Stil:
- Nur Arrays im Code.
- Nur Terminalausgabe (print), keine Dateien/Tabellen.
- Einheiten: f in Millimetern (mm), D in dpt.

So nutzen:
1) Tragen Sie unten Ihre Brennweiten f_i (mm) ein.
2) Tragen Sie die zugehörigen Unsicherheiten _f_i (mm) ein.
   (z. B. aus Teil 1a/1b: nehmen Sie für jede Messung die dort ermittelte _f,
```

```
    oder für den Gesamtwert die kombinierte  _f_final.)
3) Ausführen - das Skript berechnet D_i,  _D_i, sowie einen Gesamtwert.
"""

# =========================
# EINGABEDATEN (mm)
# =========================
# Beispielwerte - ERSETZEN durch Ihre f_i und  _f_i (jeweils gleiche Länge).
# Sie können hier z. B. die einzelnen f_i aus Teil 1a einsetzen (und dortige _f(instr) je Messung),
# oder NUR einen Gesamtwert (dann Arrays der Länge 1).
f_mm       = [112.6, 160.0]      # Beispiel: Mittel aus 1a (~112.6 mm) und aus 1b (~160 mm)
sigma_f_mm = [0.5,   30.0]       # Beispiel:  _f_final aus 1a (~0.5 mm) und 1b (~30 mm)

# =========================
# HILFSFUNKTIONEN
# =========================
def dioptrien_aus_f_mm(f_mm: float, sigma_f_mm: float) -> Tuple[float, float]:
    """
    D = 1000 / f_mm  (weil f in mm)
     _D = 1000 *  _f_mm / f_mm^2
    """
    D = 1000.0 / f_mm
    sigma_D = 1000.0 * sigma_f_mm / (f_mm**2)
    return D, sigma_D

def fmt_pm(val: float, unc: float, einheit: str) -> str:
    if unc <= 0 or not math.isfinite(unc):
        return f"{val:.3g} {einheit}"
    e = int(math.floor(math.log10(abs(unc)))) if unc != 0 else 0
    lead = round(unc/(10**e))
    n_sig = 2 if lead == 1 else 1
    prec = -e + (n_sig - 1)
    val_r = round(val, prec)
    unc_r = round(unc, prec)
    return f"{val_r:.{max(0,prec)}f} ± {unc_r:.{max(0,prec)}f} {einheit}"

def kombi_sigma(stat: float, instr: float, mode: str="max") -> float:
    return max(stat, instr) if mode == "max" else math.hypot(stat, instr)

# =========================
# RECHNUNG
# =========================
print("\n=== TEIL 3 - DIOPTRISCHE STÄRKE (D = 1/f) ===")
```

```python
assert len(f_mm) == len(sigma_f_mm) and len(f_mm) >= 1, "Arrays f_mm und␣
 ↪sigma_f_mm müssen gleich lang und nicht leer sein."

# Einzelergebnisse
D_vals, sD_vals = [], []
for i, (f_i, sf_i) in enumerate(zip(f_mm, sigma_f_mm), start=1):
    D_i, sD_i = dioptrien_aus_f_mm(f_i, sf_i)
    D_vals.append(D_i); sD_vals.append(sD_i)
    print(f"[Messung {i}] f={f_i:.3f} mm,  _f={sf_i:.3f} mm  ->  D={D_i:.3f}␣
 ↪dpt,  _D={sD_i:.3f} dpt")

# Gesamtwert-Variante A (empfohlen): aus dem GEMITTELTEN f und einer␣
 ↪kombinierten  _f
f_mean = float(np.mean(f_mm))
f_stat = float(np.std(f_mm, ddof=1)) if len(f_mm) >= 2 else float(sigma_f_mm[0])
f_instr = float(np.mean(sigma_f_mm))
f_sigma = kombi_sigma(f_stat, f_instr, mode="max")
D_mean, D_sigma = dioptrien_aus_f_mm(f_mean, f_sigma)

print("\nGESAMT (aus f und kombinierter  _f):")
print("  f  =", fmt_pm(f_mean, f_sigma, "mm"))
print("  D  =", fmt_pm(D_mean, D_sigma, "dpt"))
print(f"  (Stat.)  _f = {f_stat:.3g} mm; (Instr./repr.)  _f = {f_instr:.3g} mm; ␣
 ↪End- _f = {f_sigma:.3g} mm")

# Gesamtwert-Variante B (optional): gewichtetes Mittel der D_i mit Gewichten 1/
 ↪ _D_i^2
# (nur sinnvoll, wenn mehrere unabhängige D_i mit verschiedenen  _D_i vorliegen)
if len(D_vals) >= 2 and all(s > 0 for s in sD_vals):
    w = np.array([1.0/(s*s) for s in sD_vals])
    D_wmean = float(np.sum(w * np.array(D_vals)) / np.sum(w))
    D_wsigma = float(1.0 / math.sqrt(np.sum(w)))
    print("\nOptional: gewichtetes Mittel über D_i (Gewichte = 1/ _D_i²):")
    print("  D_w =", fmt_pm(D_wmean, D_wsigma, "dpt"))

print("\nHinweis:")
print("- Wenn Sie nur EINEN finalen f-Wert (z. B. aus 1a oder 1b) verwenden,␣
 ↪tragen Sie einfach ein Paar in die Arrays ein.")
print("-  _D wird korrekt aus  _f mit D = 1000/f_mm propagiert (mm → dpt).")
```

```
=== TEIL 3 - DIOPTRISCHE STÄRKE (D = 1/f) ===
[Messung 1] f=112.600 mm,  _f=0.500 mm  ->  D=8.881 dpt,  _D=0.039 dpt
[Messung 2] f=160.000 mm,  _f=30.000 mm  ->  D=6.250 dpt,  _D=1.172 dpt

GESAMT (aus f und kombinierter  _f):
  f  = 140 ± 30 mm
```

```
  D  = 7 ± 2 dpt
  (Stat.)  _f = 33.5 mm; (Instr./repr.)  _f = 15.2 mm;  End- _f = 33.5 mm

Optional: gewichtetes Mittel über D_i (Gewichte = 1/ _D_i²):
  D_w = 8.88 ± 0.04 dpt

Hinweis:
- Wenn Sie nur EINEN finalen f-Wert (z. B. aus 1a oder 1b) verwenden, tragen Sie
einfach ein Paar in die Arrays ein.
-  _D wird korrekt aus  _f mit D = 1000/f_mm propagiert (mm → dpt).
```

Interpretation: Positive $D \rightarrow$ Sammellinse, negative $D \rightarrow$ Zerstreuungslinse.

1.2 Experiment 2: Linsenfehler

1.2.1 Grundlagen

Linsenfehler entstehen durch Abweichungen idealer Abbildung:

- **Sphärische Aberration:** achsferne Strahlen haben einen näheren Brennpunkt als achsennahe

- **Chromatische Aberration:** aufgrund der Dispersion wird blaues Licht stärker gebrochen als rotes

- **Astigmatismus:** unterschiedliche Krümmung in verschiedenen Ebenen führt zu Linien- statt Punktbildern

1.2.2 Versuchsaufbau, Durchführung und Auswertung

2.3.1 Sphärische Aberration

1. Verwende denselben Aufbau wie in Experiment 1 (1a).

2. Bringe abwechselnd **zwei Blenden** an, um achsenferne bzw. achsennahe Strahlen zu isolieren.

3. Miss die Bildweiten b_{nah} und b_{fern}.

4. Vergleiche die Werte im Rahmen der Messgenauigkeit und beurteile, welche größer ist.

2.3.2 Chromatische Aberration

1. Entferne den Gegenstand, vergrößere den Abstand zwischen Lichtquelle und Linse auf 1 m, um **annähernd paralleles Licht** zu erzeugen.

2. Verwende nacheinander die **Farbfilter** (rot, grün, blau) und finde jeweils die Position des scharfen Bildes.

3. Miss die entsprechenden Bildweiten und vergleiche ihre Reihenfolge (blau $<$ grün $<$ rot erwartet).

4. Bestimme Messunsicherheiten analog zu Teil 1.

[18]:
```
# =====================================================================
# TEIL 4.1 - SPHÄRISCHE ABERRATION
# Vorgehen je Messung:
#   Fixe Linse und Objekt (g bekannt); Schirm auf Schärfe:
#   - nur AXIALSTRAHLEN (Blende Mitte): x_S_ax
#   - nur RANDSTRAHLEN (Blende Rand):    x_S_rand
#   b_ax = |x_S_ax - x_L|, b_rand = |x_S_rand - x_L|
#   Vergleich von b und/oder der resultierenden f = g*b/(g+b)
# =====================================================================

print("\n=== TEIL 4.1 - SPHÄRISCHE ABERRATION ===")

# -------------------------
# EINGABEDATEN TEIL 4.1 (mm)
# -------------------------
# Mehrere Messungen möglich:
x_G_41   = [1601.0, 1601.0, 1601.0]
x_L_41   = [1300.0, 1300.0, 1300.0]
x_S_ax   = [990.0, 990.0, 988.5]  # Schirmposition bei Schärfe (Axialstrahlen)
x_S_rand = [1038.0, 1040.0, 1038.0]  # Schirmposition bei Schärfe (Randstrahlen)

N41 = len(x_G_41)
assert all(len(arr)==N41 for arr in [x_L_41, x_S_ax, x_S_rand]), "Array-Längen␣
↪in TEIL 4.1 ungleich."

# Unsicherheiten:
sigma_xG = SIGMA_LESEN
sigma_xL = math.hypot(SIGMA_LESEN, 0.5*SIGMA_FOKUS)
sigma_xS_ax = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)
sigma_xS_rand = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)

sigma_g = diff_sigma(sigma_xL, sigma_xG)
sigma_b_ax = diff_sigma(sigma_xS_ax,   sigma_xL)
sigma_b_rand = diff_sigma(sigma_xS_rand, sigma_xL)

delta_b_vals = []
sigma_delta_b_vals = []

f_ax_vals, f_rand_vals = [], []
sigma_f_ax_instr, sigma_f_rand_instr = [], []

for i in range(N41):
    g = abs(x_L_41[i] - x_G_41[i])
    b_ax = abs(x_S_ax[i]  - x_L_41[i])
    b_rd = abs(x_S_rand[i]- x_L_41[i])
```

```python
    delta_b = b_ax - b_rd
    sigma_delta_b = math.hypot(sigma_b_ax, sigma_b_rand)

    f_ax = f_aus_gb(g, b_ax)
    f_rd = f_aus_gb(g, b_rd)
    sf_ax = sigma_f_aus_gb(g, b_ax, sigma_g, sigma_b_ax)
    sf_rd = sigma_f_aus_gb(g, b_rd, sigma_g, sigma_b_rand)

    delta_b_vals.append(delta_b)
    sigma_delta_b_vals.append(sigma_delta_b)
    f_ax_vals.append(f_ax)
    f_rand_vals.append(f_rd)
    sigma_f_ax_instr.append(sf_ax)
    sigma_f_rand_instr.append(sf_rd)

    print(f"[Messung {i+1}] g={g:.2f} mm, b_ax={b_ax:.2f} mm, b_rand={b_rd:.2f} mm -> Δb={delta_b:.2f} ± {sigma_delta_b:.2f} mm")
    print(f"    f(ax)={f_ax:.3f} mm ( {sf_ax:.3f}),  f(rand)={f_rd:.3f} mm ( {sf_rd:.3f})")

# Mittelwerte/Beurteilung
delta_b_mean = float(np.mean(delta_b_vals))
sigma_delta_b_mean = float(np.mean(sigma_delta_b_vals))
f_ax_mean = float(np.mean(f_ax_vals))
f_rd_mean = float(np.mean(f_rand_vals))

# Endunsicherheit für f-Werte (hier exemplarisch als Mittel der instrumentellen, da Fokus dominiert)
f_ax_sigma_final = float(np.mean(sigma_f_ax_instr))
f_rd_sigma_final = float(np.mean(sigma_f_rand_instr))

print("\nERGEBNIS TEIL 4.1:")
print(f"  Δb̄ = {fmt_pm(delta_b_mean, sigma_delta_b_mean, 'mm')}  (positiv erwartet: Fokus Axial > Rand)")
print(f"  f̄(ax)  = {fmt_pm(f_ax_mean,  f_ax_sigma_final,  'mm')}")
print(f"  f̄(rand)= {fmt_pm(f_rd_mean,  f_rd_sigma_final,  'mm')}")
print("  Bewertung: Signifikant, wenn |Δb| > ~2· (Δb).")

# ======================================================================
# TEIL 4.2 - CHROMATISCHE ABERRATION (R/G/B, quasi-paralleler Strahl)
# Vorgehen:
#   Große Gegenstandsweite (  parallel) → effektive Brennweite f( )   b( ) = |x_S( ) - x_L|
#   Für jede Farbe Schärfe suchen und Schirmposition notieren.
# ======================================================================
```

```
print("\n=== TEIL 4.2 - CHROMATISCHE ABERRATION ===")

# -------------------------
# EINGABEDATEN TEIL 4.2 (mm)
# -------------------------
# Mehrere Wiederholungen möglich; Linse fest, Schirm je Farbe separat␣
↪fokussiert.
x_L_42 = [650.0, 800.0, 950.0]      # Linsenposition
x_S_R  = [471.0, 606.0, 751.0]      # Schirm Rot
x_S_G  = [466.0, 609.0, 752.0]      # Schirm Grün
x_S_B  = [461.0, 611.0, 751.5]      # Schirm Blau

N42 = len(x_L_42)
assert all(len(arr)==N42 for arr in [x_S_R, x_S_G, x_S_B]), "Array-Längen in␣
↪TEIL 4.2 ungleich."

sigma_xL = SIGMA_LESEN
sigma_xS = math.hypot(SIGMA_LESEN, SIGMA_FOKUS)  # Fokussieren je Farbe

sigma_b = diff_sigma(sigma_xS, sigma_xL)        # für alle Farben ähnlich

f_R_vals, f_G_vals, f_B_vals = [], [], []

for i in range(N42):
    bR = abs(x_S_R[i] - x_L_42[i])
    bG = abs(x_S_G[i] - x_L_42[i])
    bB = abs(x_S_B[i] - x_L_42[i])

    # Bei quasi-parallelem Strahl: f( )   b( )
    fR, fG, fB = bR, bG, bB
    f_R_vals.append(fR); f_G_vals.append(fG); f_B_vals.append(fB)

    print(f"[Wdh. {i+1}] b_R={bR:.2f} mm, b_G={bG:.2f} mm, b_B={bB:.2f} mm  ->␣
↪f_R {fR:.2f}, f_G {fG:.2f}, f_B {fB:.2f} mm")

fR_mean, fG_mean, fB_mean = map(float, [np.mean(f_R_vals), np.mean(f_G_vals),␣
↪np.mean(f_B_vals)])
# Instrumentelle Unsicherheit ~ _b; statistisch (Streuung) hier klein - nehmen␣
↪wir _final  max( _b, s_stat)
fR_stat = float(np.std(f_R_vals, ddof=1)) if N42>=2 else sigma_b
fG_stat = float(np.std(f_G_vals, ddof=1)) if N42>=2 else sigma_b
fB_stat = float(np.std(f_B_vals, ddof=1)) if N42>=2 else sigma_b

sigma_fR = max(sigma_b, fR_stat)
sigma_fG = max(sigma_b, fG_stat)
sigma_fB = max(sigma_b, fB_stat)
```

```
print("\nERGEBNIS TEIL 4.2:")
print(f"  f(R) = {fmt_pm(fR_mean, sigma_fR, 'mm')}")
print(f"  f(G) = {fmt_pm(fG_mean, sigma_fG, 'mm')}")
print(f"  f(B) = {fmt_pm(fB_mean, sigma_fB, 'mm')}")
print("  Erwartung: f(B) < f(G) < f(R) (stärkeres Brechen im Blauen → kürzere␣
↪Brennweite).")
```

```
=== TEIL 4.1 - SPHÄRISCHE ABERRATION ===
[Messung 1] g=301.00 mm, b_ax=310.00 mm, b_rand=262.00 mm -> Δb=48.00 ± 1.87 mm
    f(ax)=152.717 mm ( 0.391),  f(rand)=140.075 mm ( 0.422)
[Messung 2] g=301.00 mm, b_ax=310.00 mm, b_rand=260.00 mm -> Δb=50.00 ± 1.87 mm
    f(ax)=152.717 mm ( 0.391),  f(rand)=139.501 mm ( 0.424)
[Messung 3] g=301.00 mm, b_ax=311.50 mm, b_rand=262.00 mm -> Δb=49.50 ± 1.87 mm
    f(ax)=153.080 mm ( 0.390),  f(rand)=140.075 mm ( 0.422)

ERGEBNIS TEIL 4.1:
  Δb̄ = 49 ± 2 mm  (positiv erwartet: Fokus Axial > Rand)
  f̄(ax)  = 152.8 ± 0.4 mm
  f̄(rand)= 139.9 ± 0.4 mm
  Bewertung: Signifikant, wenn |Δb| > ~2· (Δb).

=== TEIL 4.2 - CHROMATISCHE ABERRATION ===
[Wdh. 1] b_R=179.00 mm, b_G=184.00 mm, b_B=189.00 mm  -> f_R 179.00, f_G 184.00,
f_B 189.00 mm
[Wdh. 2] b_R=194.00 mm, b_G=191.00 mm, b_B=189.00 mm  -> f_R 194.00, f_G 191.00,
f_B 189.00 mm
[Wdh. 3] b_R=199.00 mm, b_G=198.00 mm, b_B=198.50 mm  -> f_R 199.00, f_G 198.00,
f_B 198.50 mm

ERGEBNIS TEIL 4.2:
  f(R) = 191 ± 10 mm
  f(G) = 191 ± 7 mm
  f(B) = 192 ± 5 mm
  Erwartung: f(B) < f(G) < f(R) (stärkeres Brechen im Blauen → kürzere
Brennweite).
```