

Di01-PW7

November 18, 2025

Wärme und Wärmetransport

1 Allgemeine Grundlagen

1.1 Teilchen- und Wellenmodelle des Lichtes

Bereits Isaac Newton ordnete Licht Teilcheneigenschaften zu. In seiner Korpuskulartheorie erklärte er Farben als Ströme von Lichtteilchen, die sich mit unterschiedlichen Geschwindigkeiten bewegen und durch Kräfte abgelenkt werden können. Dies sollte Brechung, Reflexion und Beugung erklären. Allerdings konnte dieses Modell Interferenzerscheinungen nicht erklären.

Die spätere Wellentheorie von Huygens, Young und Fresnel löste das Teilchenmodell ab, da sie Interferenz und Beugung beschreiben konnte. Sie ging davon aus, dass sich Lichtwellen in einem hypothetischen Medium, dem “Lichtäther”, ausbreiten. Maxwell zeigte später, dass Licht eine elektromagnetische Transversalwelle ist.

Doch Wärme- und Temperaturstrahlung sowie die Wechselwirkung von Licht mit Materie ließen sich nicht vollständig mit dem klassischen Wellenmodell erklären. So wurde erneut der Teilchenaspekt benötigt: elektromagnetische Strahlung ist in Energiepakete quantisiert.

Planck führte das Wirkungsquantum h ein und formulierte, dass die Energie eines Strahlungsquants proportional zur Frequenz ν ist:

$$E = h \cdot \nu$$

Einstein zeigte anhand des Photoeffekts, dass Licht selbst aus Quanten besteht: Photonen. Dies begründete den Welle-Teilchen-Dualismus.

1.2 Wärme

Wärme ist eine Form von Energie, die aufgrund eines Temperaturunterschieds von einem Körper auf einen anderen übertragen wird.

Es gibt zwei Betrachtungsweisen:

- **Thermodynamik:** beschreibt makroskopische Größen wie Druck, Volumen, Temperatur oder Energie.
- **Statistische Mechanik:** betrachtet mikroskopische Teilchen und leitet aus deren Verhalten makroskopische Eigenschaften ab.

1.3 Hauptsätze der Wärmelehre

0. Befinden sich zwei Systeme jeweils im thermischen Gleichgewicht mit einem dritten System, so stehen sie auch miteinander im Gleichgewicht. Dies ermöglicht eine Definition der Temperatur.
1. Die Änderung der inneren Energie eines abgeschlossenen Systems entspricht der zugeführten Wärme plus der zugeführten Arbeit. Er ist eine Form des Energieerhaltungssatzes.
2. Wärme fließt niemals spontan von einem kälteren zu einem wärmeren Körper. Zudem ist es unmöglich, eine Maschine zu bauen, die ausschließlich Wärme aufnimmt und vollständig in Arbeit umwandelt.
3. Ein Abkühlen bis zum absoluten Nullpunkt ist in endlich vielen Schritten unmöglich.

1.4 Mechanismen des Wärmetransports

Es gibt drei Arten des Wärmetransports:

1. **Wärmeleitung**
2. **Konvektion**
3. **Wärmestrahlung**

1.4.1 Wärmeleitung

Wärmeleitung findet hauptsächlich in Festkörpern statt. Gute elektrische Leiter sind meist auch gute Wärmeleiter, da die Energieübertragung durch Leitungselektronen sehr effizient ist. In Isolatoren erfolgt die Wärmeübertragung über Gitterschwingungen, sogenannte Phononen.

1.4.2 Konvektion

Konvektion tritt hauptsächlich in Flüssigkeiten und Gasen auf. Warme Materie dehnt sich aus, wird leichter und steigt auf, während kältere nachströmt. Dieser Mechanismus ist entscheidend für Wetter, Klima, technische Heizverfahren und viele natürliche Vorgänge.

1.5 Wärmestrahlung

Wärmestrahlung ist elektromagnetische Strahlung, die durch die thermische Bewegung von Ladungen in Materie erzeugt wird. Sie kann auch im Vakuum übertragen werden.

Erhitzt man einen Körper, so emittiert er Strahlung in einem breiten Spektrum, das von der Temperatur abhängt. Ein Objekt beginnt bei hoher Temperatur sichtbar zu leuchten (z. B. glühendes Metall).

2 Wärmestrahlung – Grundlagen

2.1 Wärmestrahlung

Alle Körper mit Temperatur $T > 0$ emittieren Strahlung. Je höher die Temperatur, desto höher der Anteil kurzwelliger Strahlung. Die spektrale Intensitätsverteilung wird mit dem Modell des schwarzen Körpers beschrieben.

2.2 Der Schwarze Körper

Ein idealer schwarzer Körper absorbiert alle einfallende Strahlung und reflektiert nichts. Ein guter experimenteller Näherungswert ist ein heißer Hohlraum mit kleiner Öffnung.

Im Inneren reflektiert die Strahlung mehrfach und wird vollständig absorbiert. Die darin stehenden Wellen sind quantisierte Eigenmoden. Die austretende Strahlung ist die sogenannte Schwarzkörperstrahlung.

2.3 Kirchhoffsches Strahlungsgesetz

Das Verhältnis von Emissionsvermögen zu Absorptionsvermögen ist für alle Körper gleich. Ein Objekt strahlt umso besser, je stärker es Strahlung absorbiert. Ein schwarzer Körper hat Absorption = 1 und daher maximales Emissionsvermögen.

2.4 Plancksches Strahlungsgesetz

Die spektrale Strahldichte eines schwarzen Körpers lautet:

$$L(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{\exp\left(\frac{hc}{\lambda kT}\right) - 1}$$

Die Energiedichte im Hohlraum:

$$U(\lambda, T) = \frac{4\pi}{c} L(\lambda, T)$$

Das Stefan-Boltzmann-Gesetz ergibt sich durch Integration über alle Wellenlängen:

$$P = \sigma AT^4$$

2.5 Farbtemperatur einer Glühlampe

Für ein bestimmtes λ gilt:

$$\frac{L(\lambda, T_1)}{L(\lambda, T_2)} = \frac{I_1}{I_2}$$

Unter Annahme des Wienschen Bereichs:

$$\ln\left(\frac{I_1}{I_2}\right) = \left(\frac{1}{T_2} - \frac{1}{T_1}\right) \frac{ch}{k\lambda}$$

Die elektrische Leistung ist:

$$\frac{P_1}{P_2} = \frac{T_1^4}{T_2^4}$$

Damit können die Temperaturen der Glühwendel berechnet werden.

2.6 Auswertung

```
[7]: import numpy as np
import matplotlib.pyplot as plt

# =====
# Konstanten
# =====

h = 6.62607015e-34
c = 2.99792458e8
k_B = 1.380649e-23

lambda0 = 580e-9
C_ch_klambda = c * h / (k_B * lambda0)

# =====
# UNI-T UT61B - Gerätefehler
# (DC-Spannung und DC-Strom, nach Leitfaden)
# =====

def ut61b_dc_voltage_error(value, vrangle):
    """
    Geräteunsicherheit für Gleichspannung mit UNI-T UT61B.
    'value' in Volt (Mittelwert),
    'vrangle' gewählter Messbereich:
    0.04, 0.4, 4, 40, 400, 600 (V)

    Daten aus Leitfaden, Abschnitt 9.4 UNI-T UT61B:
    DC Voltage:
        40 mV, 400 mV : ±(0.8% vom Messwert + 3 digits)
        4 V - 400 V   : ±(0.5% vom Messwert + 1 digit)
        600 V         : ±(1.0% vom Messwert + 3 digits)
    Auflösungen:
        40 mV : 0.01 mV = 1e-5 V
        400 mV : 0.1 mV = 1e-4 V
        4 V   : 0.001 V
        40 V  : 0.01 V
        400 V : 0.1 V
        600 V : 1 V
    """
    v = abs(value)

    if vrangle == 0.04:      # 40 mV
        pct = 0.008          # 0.8 %
        digits = 3
```

```

    res = 1e-5
elif vrangle == 0.4:      # 400 mV
    pct = 0.008
    digits = 3
    res = 1e-4
elif vrangle == 4:         # 4 V
    pct = 0.005            # 0.5 %
    digits = 1
    res = 1e-3
elif vrangle == 40:        # 40 V
    pct = 0.005
    digits = 1
    res = 1e-2
elif vrangle == 400:       # 400 V
    pct = 0.005
    digits = 1
    res = 1e-1
elif vrangle == 600:       # 600 V
    pct = 0.010            # 1.0 %
    digits = 3
    res = 1.0
else:
    raise ValueError("Unbekannter UT61B-Spannungsbereich (vrangle)")

return pct * v + digits * res

def ut61b_dc_current_error(value, irange):
    """
    Geräteunsicherheit für Gleichstrom mit UNI-T UT61B.
    'value' in Ampere (Mittelwert),
    'irange' gewählter Messbereich:
    0.0004, 0.004, 0.04, 0.4, 4, 10 (A)

    Daten aus Leitfaden:
    DC Current:
    400 μA, 4000 μA : ±(1.0% + 2 digits)
    40 mA, 400 mA   : ±(1.2% + 3 digits)
    4 A, 10 A       : ±(1.5% + 3 digits)
    Auflösungen:
    400 μA : 0.1 μA = 1e-7 A
    4000 μA : 0.1 μA = 1e-7 A
    40 mA   : 0.01 mA = 1e-5 A
    400 mA  : 0.1 mA  = 1e-4 A
    4 A     : 0.001 A
    10 A   : 0.01 A
    """

```

```

i = abs(value)

if irange == 0.0004:          # 400 μA
    pct = 0.010
    digits = 2
    res = 1e-7
elif irange == 0.004:         # 4000 μA
    pct = 0.010
    digits = 2
    res = 1e-7
elif irange == 0.04:          # 40 mA
    pct = 0.012
    digits = 3
    res = 1e-5
elif irange == 0.4:           # 400 mA
    pct = 0.012
    digits = 3
    res = 1e-4
elif irange == 4:              # 4 A
    pct = 0.015
    digits = 3
    res = 1e-3
elif irange == 10:             # 10 A
    pct = 0.015
    digits = 3
    res = 1e-2
else:
    raise ValueError("Unbekannter UT61B-Strombereich (irange)")

return pct * i + digits * res

# =====
# Fits, Chi2, R2
# =====

def compute_chi2_and_r2(x, y, fit_func):
    y_fit = fit_func(x)
    residuals = y - y_fit

    sigma = np.std(residuals, ddof=1)
    if sigma == 0:
        sigma = 1e-12

    chi2 = np.sum((residuals / sigma) ** 2)
    dof = len(y) - 2          # 2 Fitparameter
    chi2_dof = chi2 / dof

```

```

ss_tot = np.sum((y - np.mean(y)) ** 2)
ss_res = np.sum(residuals ** 2)
R2 = 1 - ss_res / ss_tot

return chi2, chi2_dof, R2

def fit_U_vs_inv_r2(r_mm, U_photo, label=""):
    r_m = r_mm / 1000.0
    x = 1.0 / (r_m ** 2)
    y = U_photo

    coeffs, cov = np.polyfit(x, y, deg=1, cov=True)
    a, b = coeffs
    a_err = np.sqrt(cov[0, 0])
    b_err = np.sqrt(cov[1, 1])

    def fit_func(x_val):
        return a * x_val + b

    chi2, chi2_dof, R2 = compute_chi2_and_r2(x, y, fit_func)

    print(f"Fit {label}: U = a * (1/r²) + b")
    print(f"  a = ({a:.6e} ± {a_err:.6e}) V m²")
    print(f"  b = ({b:.6e} ± {b_err:.6e}) V \n")

    return a, a_err, fit_func, x, y, (chi2, chi2_dof, R2)

# =====
# Leistung mit Unsicherheit (Leitfaden + UT61B)
# =====

def compute_power_with_errors(U_lamp, I_lamp, U_range, I_range, label=""):
    """
    U_lamp, I_lamp: Arrays (oder scalars) mit Spannungs- und Stromwerten.
    U_range: verwendeter UT61B-Spannungsbereich (z.B. 4, 40, 400)
    I_range: verwendeter UT61B-Strombereich (z.B. 0.4, 4, 10)

    Rückgabe: P_mean, DeltaP  (DeltaP nach Gauss + Leitfaden-Regel)
    """
    U_lamp = np.asarray(U_lamp, dtype=float)
    I_lamp = np.asarray(I_lamp, dtype=float)

    # Einzel-Leistungswerte
    P = U_lamp * I_lamp

```

```

P_mean = np.mean(P)

# --- Statistische Unsicherheiten von U und I (Standardfehler des Mittelwertes)
N = U_lamp.size

if N > 1:
    s_U = np.std(U_lamp, ddof=1)
    s_I = np.std(I_lamp, ddof=1)
    u_U = s_U / np.sqrt(N)
    u_I = s_I / np.sqrt(N)
else:
    u_U = 0.0
    u_I = 0.0

U_mean = np.mean(U_lamp)
I_mean = np.mean(I_lamp)

# --- Gerätfehler des UT61B (Typ-B-Unsicherheit)
dU_geraet = ut61b_dc_voltage_error(U_mean, U_range)
dI_geraet = ut61b_dc_current_error(I_mean, I_range)

# --- Leitfaden-Regel: Delta = max(ux, Delta_geraet)
dU = max(abs(u_U), abs(dU_geraet))
dI = max(abs(u_I), abs(dI_geraet))

# --- Gauss'sche Fehlerfortpflanzung für P = U * I (relativ)
relU = dU / abs(U_mean) if U_mean != 0 else 0.0
relI = dI / abs(I_mean) if I_mean != 0 else 0.0
relP = np.sqrt(relU**2 + relI**2)
dP = abs(P_mean) * relP

print(f"Leistung {label}:")
print(f"  U_mean = ({U_mean:.4f} V ± {dU:.4f}) V")
print(f"  I_mean = ({I_mean:.4f} A ± {dI:.4f}) A")
print(f"  P      = ({P_mean:.4f} W ± {dP:.4f}) W\n")

return P_mean, dP

# =====
# Temperaturen + Fehlerfortpflanzung
# =====

def compute_temperatures(I_ratio, P1, P2):
    R_P = P1 / P2
    R_I = I_ratio

```

```

R_T = R_P ** 0.25

numerator = C_ch_klambda * (1.0 - 1.0 / R_T)
denominator = np.log(R_I)

T2 = numerator / denominator
T1 = T2 * R_T

return T1, T2

def propagate_error_temperatures(I_ratio, I_ratio_err, P1, P1_err, P2, P2_err):
    T1_0, T2_0 = compute_temperatures(I_ratio, P1, P2)

    def derivative(var_name):
        if var_name == "I":
            x0, sx = I_ratio, I_ratio_err
            def f(x): return compute_temperatures(x, P1, P2)
        elif var_name == "P1":
            x0, sx = P1, P1_err
            def f(x): return compute_temperatures(I_ratio, x, P2)
        elif var_name == "P2":
            x0, sx = P2, P2_err
            def f(x): return compute_temperatures(I_ratio, P1, x)
        else:
            raise ValueError

        if sx == 0:
            sx = max(abs(x0) * 1e-4, 1e-8)

        T1_plus, T2_plus = f(x0 + sx)
        T1_minus, T2_minus = f(x0 - sx)

        dT1 = (T1_plus - T1_minus) / (2 * sx)
        dT2 = (T2_plus - T2_minus) / (2 * sx)

    return dT1, dT2

dT1_dI, dT2_dI = derivative("I")
dT1_dP1, dT2_dP1 = derivative("P1")
dT1_dP2, dT2_dP2 = derivative("P2")

T1_err = np.sqrt((dT1_dI*I_ratio_err)**2 +
                  (dT1_dP1*P1_err)**2 +
                  (dT1_dP2*P2_err)**2)

T2_err = np.sqrt((dT2_dI*I_ratio_err)**2 +

```

```

        (dT2_dP1*P1_err)**2 +
        (dT2_dP2*P2_err)**2)

    return T1_0, T1_err, T2_0, T2_err

# =====
# Plot mit Chi2/DoF und R2 im Diagramm + UT61B-Fehler für U_photo
# =====

def plot_results(x1, U1, fit1, stats1,
                  x2, U2, fit2, stats2,
                  label1, label2):
    chi2_1, chi2_dof_1, R2_1 = stats1
    chi2_2, chi2_dof_2, R2_2 = stats2

    # Aangenommener DC-Spannungsbereich für Photospannung am UT61B:
    # Hier z.B. 0.4 V (400 mV). Anpassen, falls ihr anderen Bereich hattet.
    Uphoto_range = 0.4

    U1_err = np.array([ut61b_dc_voltage_error(u, Uphoto_range) for u in U1])
    U2_err = np.array([ut61b_dc_voltage_error(u, Uphoto_range) for u in U2])

    plt.figure(figsize=(8, 6))

    plt.errorbar(x1, U1, yerr=U1_err, fmt="o", capsize=4,
                  label=f"Messung {label1}")
    plt.errorbar(x2, U2, yerr=U2_err, fmt="s", capsize=4,
                  label=f"Messung {label2}")

    x_all = np.linspace(0, max(np.max(x1), np.max(x2))*1.05, 300)
    plt.plot(x_all, fit1(x_all), label=f"Fit {label1}")
    plt.plot(x_all, fit2(x_all), label=f"Fit {label2}")

    textstr = (
        f"{label1}:\n"
        f"  Chi2      = {chi2_1:.3f}\n"
        f"  Chi2/DoF   = {chi2_dof_1:.3f}\n"
        f"  R2       = {R2_1:.5f}\n\n"
        f"{label2}:\n"
        f"  Chi2      = {chi2_2:.3f}\n"
        f"  Chi2/DoF   = {chi2_dof_2:.3f}\n"
        f"  R2       = {R2_2:.5f}"
    )

    ax = plt.gca()
    ax.text(

```

```

    0.02, 0.98, textstr,
    transform=ax.transAxes,
    va="top",
    fontsize=10,
    bbox=dict(boxstyle="round", fc="white", ec="black", alpha=0.8),
)

plt.xlabel(r"\frac{1}{r^2} [ \frac{1}{m^2}]")
plt.ylabel(r"U_{\mathrm{photo}} [V]")
plt.title(r"U vs. $r^{-2}$")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

def main():
    # ----- Bedingung 1 -----
    r1_mm = np.array([500, 543, 598, 674, 730, 791, 900, 1000, 1570], dtype=float)
    U1_photo = np.array([0.0356, 0.0333, 0.0282, 0.0243, 0.0228, 0.0206, 0.0186, 0.017, 0.0136], dtype=float)

    # Lampenspannung/-strom Messreihe
    U1_lamp = np.full_like(r1_mm, 6.00, dtype=float)
    I1_lamp = np.array([4.48, 4.532, 4.44, 4.492, 4.558, 4.495, 4.539, 4.497, 4.543], dtype=float)

    # UT61B-Bereiche für Lampenmessung (anpassen falls nötig!)
    U1_range = 0.4      # z.B. 40 V Bereich
    I1_range = 10       # z.B. 10 A Bereich

    label1 = "6 Volt"

    # ----- Bedingung 2 -----
    r2_mm = np.array([500, 543, 598, 674, 730, 791, 900, 1000, 1570], dtype=float)
    U2_photo = np.array([0.0172, 0.0161, 0.0151, 0.014, 0.0136, 0.0132, 0.0127, 0.0125, 0.0124], dtype=float)

    U2_lamp = np.full_like(r2_mm, 4.00, dtype=float)
    I2_lamp = np.array([3.618, 3.616, 3.619, 3.621, 3.619, 3.618, 3.617, 3.616, 3.615], dtype=float)

    U2_range = 0.4      # 40 V Bereich
    I2_range = 10       # 4 A Bereich

```

```

label2 = "4 Volt"

# ====== Fits ======
a1, a1_err, fit1, x1, y1, stats1 = fit_U_vs_inv_r2(r1_mm, U1_photo, label1)
a2, a2_err, fit2, x2, y2, stats2 = fit_U_vs_inv_r2(r2_mm, U2_photo, label2)

# Verhältnis der Lichtstärken
I_ratio = a1 / a2
I_ratio_err = np.sqrt((1/a2)**2 * a1_err**2 + (a1/(a2**2))**2 * a2_err**2)
print(f"I1/I2 = {I_ratio:.6f} ± {I_ratio_err:.6f}\n")

# ====== Leistungen (mit UT61B-Unsicherheiten) ======
P1, P1_err = compute_power_with_errors(U1_lamp, I1_lamp, U1_range, ↵
I1_range, label1)
P2, P2_err = compute_power_with_errors(U2_lamp, I2_lamp, U2_range, ↵
I2_range, label2)

# ====== Temperaturen ======
T1, T1_err, T2, T2_err = propagate_error_temperatures(
    I_ratio, I_ratio_err, P1, P1_err, P2, P2_err
)

print("Strahlungstemperaturen:")
print(f"  T1 ({label1}) = {T1:.2f} ± {T1_err:.2f} K")
print(f"  T2 ({label2}) = {T2:.2f} ± {T2_err:.2f} K\n")

# ====== Plot ======
plot_results(x1, y1, fit1, stats1, x2, y2, fit2, stats2, label1, label2)

if __name__ == "__main__":
    main()

```

Fit 6 Volt: $U = a \cdot (1/r^2) + b$
 $a = (6.327289e-03 \pm 1.531384e-04) \text{ V m}^2$
 $b = (1.076865e-02 \pm 3.570778e-04) \text{ V}$

Fit 4 Volt: $U = a \cdot (1/r^2) + b$
 $a = (1.430128e-03 \pm 1.038219e-04) \text{ V m}^2$
 $b = (1.114850e-02 \pm 2.420849e-04) \text{ V}$

$I1/I2 = 4.424282 \pm 0.338566$

Leistung 6 Volt:
 $U_{\text{mean}} = (6.0000 \text{ V} \pm 0.0483) \text{ V}$
 $I_{\text{mean}} = (4.5084 \text{ A} \pm 0.0976) \text{ A}$

$$P = (27.0507 \text{ W} \pm 0.6249) \text{ W}$$

Leistung 4 Volt:

$$U_{\text{mean}} = (4.0000 \text{ V} \pm 0.0323) \text{ V}$$

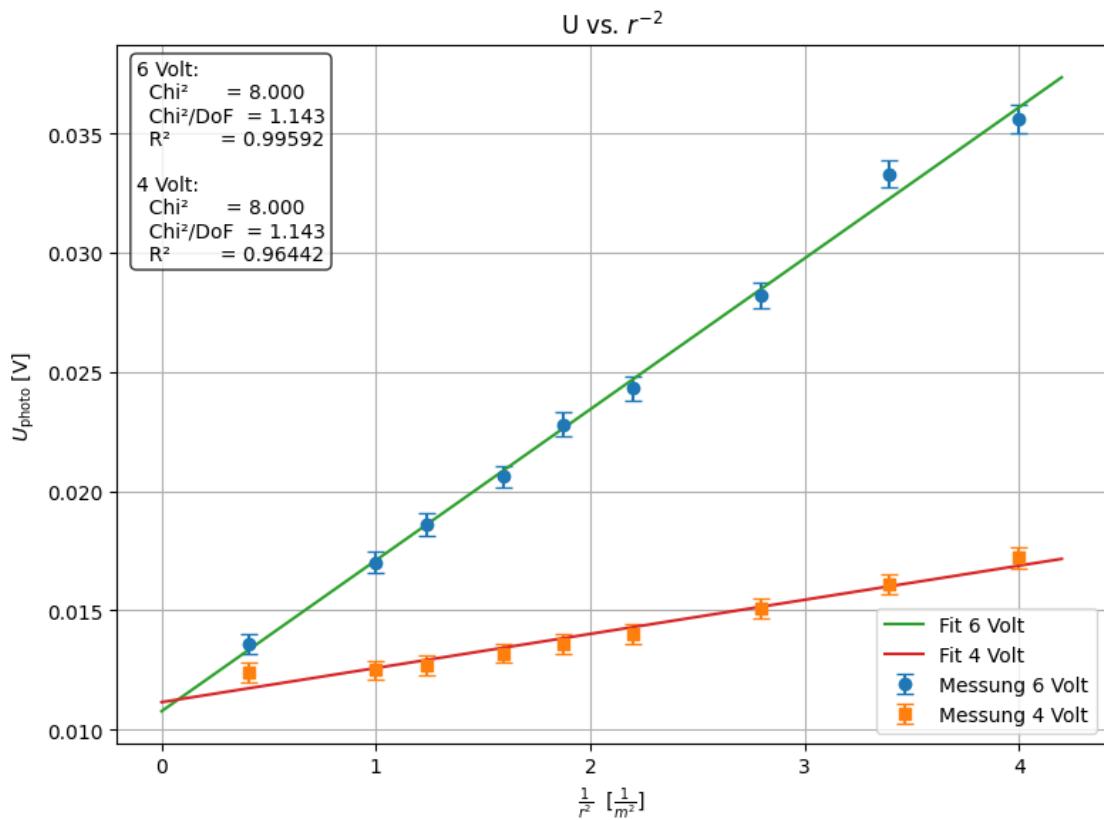
$$I_{\text{mean}} = (3.6177 \text{ A} \pm 0.0843) \text{ A}$$

$$P = (14.4707 \text{ W} \pm 0.3567) \text{ W}$$

Strahlungstemperaturen:

$$T_1 \text{ (6 Volt)} = 2823.94 \pm 220.53 \text{ K}$$

$$T_2 \text{ (4 Volt)} = 2415.09 \pm 173.89 \text{ K}$$



3 Wärmeleitung – Grundlagen

3.1 Wärmeleitung als Form des Wärmetransports

Wärme kann durch Leitung, Konvektion und Strahlung uebertragen werden. In diesem Experiment wird ausschliesslich die Wärmeleitung untersucht. Dabei wird Energie innerhalb eines festen Koerpers transportiert, ohne dass sich Materie makroskopisch bewegt.

3.2 Temperatur, Wärmestrom und Wärmeflussdichte

Der Wärmestrom ist definiert als:

$$\dot{Q} = \frac{dQ}{dt}.$$

Die Wärmeflussdichte ist der Wärmestrom pro Fläche:

$$j_q = \frac{\dot{Q}}{A}.$$

3.3 Fouriersches Gesetz

Das zentrale Gesetz fuer Wärmeleitung lautet:

$$\vec{j}_q = -\lambda \nabla T.$$

In einem eindimensionalen Stab ergibt sich:

$$j_q = -\lambda \frac{dT}{dx}.$$

Der Wärmestrom wird dann zu:

$$\dot{Q} = -\lambda A \frac{dT}{dx}.$$

3.4 Stationärer Zustand

Erhitzt man Ende des Stabes und kuehlt das andere, stellt sich nach einiger Zeit ein stationärer Zustand ein:

$$\frac{\partial T}{\partial t} = 0.$$

Damit folgt aus der Wärmeleitungsgleichung:

$$\frac{d^2T}{dx^2} = 0.$$

Die Temperaturverteilung ist daher linear:

$$T(x) = T_0 + mx.$$

Der Temperaturgradient ist konstant:

$$\frac{dT}{dx} = m.$$

3.5 Bestimmung der thermischen Leitfähigkeit

Im stationären Zustand gilt:

$$\dot{Q} = -\lambda A m.$$

Fuer die thermische Leitfähigkeit ergibt sich:

$$\lambda = -\frac{\dot{Q}}{A} \left(\frac{dT}{dx} \right)^{-1}.$$

Die relevanten Groessen sind:

- $\dot{Q} \approx P = U \cdot I$
 - A : Querschnittsfläche des Stabes
 - $\frac{dT}{dx}$: Temperaturgradient aus linearer Regression
-

3.6 Allgemeine Wärmeleitungsgleichung

Die zeitabhängige Gleichung lautet:

$$\rho c \frac{\partial T}{\partial t} = \lambda \nabla^2 T.$$

Die thermische Diffusivität ist:

$$\alpha = \frac{\lambda}{\rho c}.$$

Im stationären Fall folgt erneut:

$$\nabla^2 T = 0.$$

```
[24]: import os

# =====#
# CSV LOADING
# =====#

def load_csv_or_default(csv_path, default_dict):
    """
    Laedt CSV, wenn vorhanden. Wenn nicht, werden Default-Daten genutzt.
    """
```

```

Default-Daten kommen in einem Dictionary:
    { "x_mm": np.array([...]),
      "T_C": np.array([...]),
      "U_heiz": np.array([...]),
      "I_heiz": np.array([...]) }

"""

if csv_path is None or not os.path.exists(csv_path):
    print("Keine CSV angegeben -> benutze eingebaute Daten.\n")
    return default_dict

print(f"Lade CSV: {csv_path}\n")

data = np.genfromtxt(csv_path, delimiter=",", names=True, encoding="utf8")

result = {}
for key in default_dict:
    if key in data.dtype.names:
        result[key] = data[key]

return result

# =====
# Fluke 179 - Geraetefehler
# =====

def fluke179_dc_voltage_error(value, vrangle):
    v = abs(value)
    if vrangle == 0.6:
        return 0.0009 * v + 2 * 1e-4
    elif vrangle == 6:
        return 0.0009 * v + 2 * 1e-3
    elif vrangle == 60:
        return 0.0009 * v + 2 * 1e-2
    elif vrangle == 600:
        return 0.0009 * v + 2 * 1e-1
    elif vrangle == 1000:
        return 0.0015 * v + 2 * 1.0
    else:
        raise ValueError("Unbekannter Spannungsbereich")

def fluke179_dc_current_error(value, irange):
    i = abs(value)
    if irange == 0.06:
        return 0.01 * i + 3 * 1e-5
    elif irange == 0.4:
        return 0.01 * i + 3 * 1e-4

```

```

    elif irange == 6:
        return 0.01 * i + 3 * 1e-3
    elif irange == 10:
        return 0.01 * i + 3 * 1e-2
    else:
        raise ValueError("Unbekannter Strombereich")

# =====
# Fit / Chi2 / R2
# =====

def compute_chi2_and_r2(x, y, fit_func):
    y_fit = fit_func(x)
    residuals = y - y_fit
    sigma = np.std(residuals, ddof=1) or 1e-12
    chi2 = np.sum((residuals / sigma) ** 2)
    dof = len(y) - 2
    chi2_dof = chi2 / dof
    ss_tot = np.sum((y - np.mean(y)) ** 2)
    ss_res = np.sum(residuals ** 2)
    R2 = 1 - ss_res / ss_tot
    return chi2, chi2_dof, R2

def fit_T_vs_x(x_mm, T, l, label=""):
    x_m = x_mm * l / 94.0
    y = T
    coeffs, cov = np.polyfit(x_m, y, 1, cov=True)
    m, b = coeffs
    m_err, b_err = np.sqrt(cov[0,0]), np.sqrt(cov[1,1])

    def f(x):
        return m * x + b

    chi2, chi2_dof, R2 = compute_chi2_and_r2(x_m, y, f)

    print(f"Fit {label}: T(x) = {m}*x + {b}")
    print(f"  m = ({m:.4e} ± {m_err:.4e}) K/m")
    print(f"  b = ({b:.4e} ± {b_err:.4e}) K\n")

    return m, m_err, f, x_m, y, (chi2, chi2_dof, R2)

# =====
# Leistung + Fehler
# =====

```

```

def compute_power_with_errors(U, I, U_range, I_range):
    U, I = np.array(U), np.array(I)
    P = U * I
    P_mean = np.mean(P)

    N = len(U)
    uU = np.std(U, ddof=1)/np.sqrt(N) if N>1 else 0
    uI = np.std(I, ddof=1)/np.sqrt(N) if N>1 else 0

    dU = max(uU, fluke179_dc_voltage_error(np.mean(U), U_range))
    dI = max(uI, fluke179_dc_current_error(np.mean(I), I_range))

    relP = np.sqrt((dU/np.mean(U))**2 + (dI/np.mean(I))**2)
    dP = abs(P_mean)*relP

    print(f"Leistung: P = ({P_mean:.3f} ± {dP:.3f}) W\n")
    return P_mean, dP

# =====
# Flaeche rechteckiges Band + Fehler
# =====

def compute_area_rectangle(b, db, h, dh):
    """
    Querschnittsflaeche eines duennen rechteckigen Bandes:
        A = b * h
    Fehler:
        dA^2 = (h * db)^2 + (b * dh)^2
    """
    A = b * h
    dA = np.sqrt((h * db)**2 + (b * dh)**2)
    return A, dA

# =====
# und Fehler
# =====

def compute_lambda(P, A, m):
    """
    lambda = - P / (A * m)
    """
    return - P / (A * m)

def propagate_error_lambda(P, dP, A, dA, m, dm):
    """
    """

```

```

Fehlerfortpflanzung fuer lambda = - P / (A m)

lambda/P = -1 / (A m)
lambda/A = P / (A^2 m)
lambda/m = P / (A m^2)

d lambda^2 = (lambda/P dP)^2 + (lambda/A dA)^2 + (lambda/m dm)^2
"""
dl_dP = -1.0 / (A * m)
dl_dA = P / (A**2 * m)
dl_dm = P / (A * m**2)

var_lambda = (dl_dP * dP)**2 + (dl_dA * dA)**2 + (dl_dm * dm)**2
return np.sqrt(var_lambda)

# =====
# Plot
# =====

def plot_results(x_m, T, T_err, fit_func, stats, lam, dlam, label):
    chi2, chi2_dof, R2 = stats

    plt.figure(figsize=(8,6))
    plt.scatter(x_m, T, marker="o", label="Messpunkte")
    x_all = np.linspace(0, np.max(x_m)*1.05, 300)
    plt.plot(x_all, fit_func(x_all), "-.", label="Fit", color="hotpink")

    txt = f"Chi^2/DoF = {chi2_dof:.3f}\nR^2 = {R2:.5f}\n"
    txt += f"lambda = {lam:.2f} ± {dlam:.2f} W/(mK)"

    plt.gca().text(0.02,0.98,txt,va="top",
                  transform=plt.gca().transAxes,
                  bbox=dict(boxstyle="round",fc="white"))

    plt.xlabel("x [m]")
    plt.ylabel("T [°C]")
    plt.title(label)
    plt.grid()
    plt.legend()
    plt.tight_layout()
    plt.show()

# =====
# MAIN
# =====

```

```

def main(csv_path=None):
    # -----
    # Default-Daten falls keine CSV vorhanden ist
    # -----
    defaults = {
        "x_mm": np.array([20,40,60,80,100,120], float),
        "T_C": np.array([60.2,55.8,51.5,47.0,42.9,39.0], float),
        "U_heiz": np.array([12.05,12.02,12.03,12.04,12.01], float),
        "I_heiz": np.array([0.950,0.948,0.951,0.949,0.950], float),
    }

    data = load_csv_or_default(csv_path, defaults)

    x_mm = data["x_mm"]
    T_C = data["T_C"]
    U = np.full_like(x_mm, 14, dtype=float)
    I = np.full_like(x_mm, 0.043, dtype=float)

    # -----
    # Geometrie: duennes rechteckiges Band
    # -----
    # Beispielwerte: Breite b und Dicke h in m
    # HIER deine realen Werte und Unsicherheiten eintragen:
    b = 0.0099
    db = 0.00005
    h = 0.004
    dh = 0.00005

    l = 0.0396

    A, dA = compute_area_rectangle(b, db, h, dh)

    print(f"Breite b = ({b:.4f} ± {db:.4f}) m")
    print(f"Dicke h = ({h:.4f} ± {dh:.4f}) m")
    print(f"Querschnittsflaeche A = ({A:.4e} ± {dA:.4e}) m^2\n")

    # Temp-Fehler (z.B. Thermoellement ±0.5 K)
    T_err = np.full_like(T_C, 0.0)

    # -----
    # Fit T(x)
    # -----
    m, dm, f, x_m, y, stats = fit_T_vs_x(x_mm, T_C, l, "")

    # -----
    # Leistung

```

```

# -----
U_range = 60
I_range = 6
P, dP = compute_power_with_errors(U, I, U_range, I_range)

# -----
# Lambda
# -----
lam = compute_lambda(P, A, m)
dlam = propagate_error_lambda(P, dP, A, dA, m, dm)

print(f"Lambda = ({lam:.2f} ± {dlam:.2f}) W/(mK)\n")

# -----
# Plot
# -----
plot_results(x_m, y, T_err, f, stats, lam, dlam, "Temperaturverlauf entlang"
             ↪des Bandes")

if __name__ == "__main__":
    # Beispiel:
    main("hitzmap.csv")
    #main(None)

```

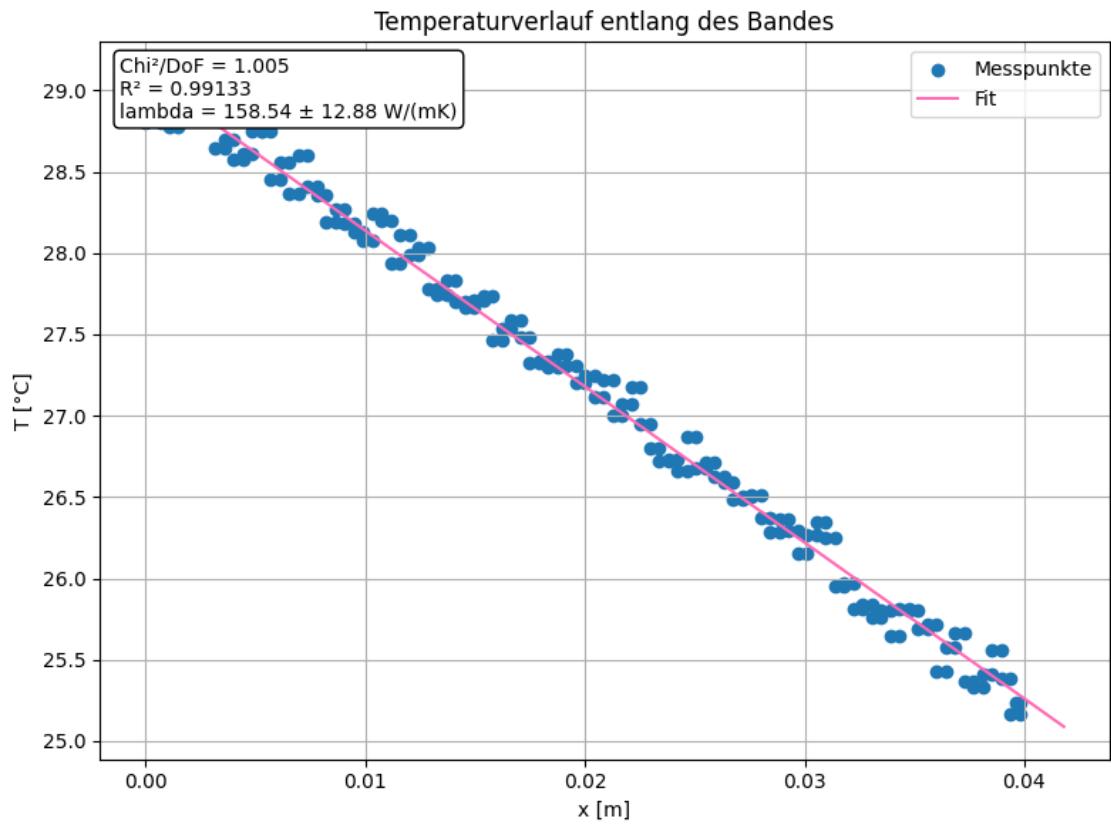
Lade CSV: hitzmap.csv

Breite b = (0.0099 ± 0.0001) m
 Dicke h = (0.0040 ± 0.0001) m
 Querschnittsfläche A = (3.9600e-05 ± 5.3388e-07) m²

Fit : T(x) = m*x + b
 m = (-9.5885e+01 ± 6.5408e-01) K/m
 b = (2.9097e+01 ± 1.5230e-02) K

Leistung: P = (0.602 ± 0.048) W

Lambda = (158.54 ± 12.88) W/(mK)



Offensichtlich, das ist Duraluminium mit Leitfähigkeit 165 W/(m K).