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:

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

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:

$$P = \sigma A T^4$$

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

[ ]:
```python
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)


# ============================================================
# Fluke 179 - Gerätefehler
# (DC-Spannung und DC-Strom)
# ============================================================

def fluke179_dc_voltage_error(value, vrange):
    """
    Geräteunsicherheit für Gleichspannung mit Fluke 179.
    'value'  in Volt (Mittelwert),
    'vrange' gewählter Messbereich: 0.6, 6, 60, 600, 1000 (V).

    Daten aus Leitfaden, Kap. 9.3: FLUKE 175/179/87V.
    Modell 179, DC:
      0.6, 6, 60, 600 V: ±(0.09% vom Messwert + 2 digits)
      1000 V           : ±(0.15% vom Messwert + 2 digits)
    Auflösungen:
      0.6 V : 0.1 mV = 1e-4 V
      6 V   : 1 mV   = 1e-3 V
      60 V  : 10 mV  = 1e-2 V
      600 V : 0.1 V  = 1e-1 V
      1000 V: 1 V
    """
    v = abs(value)

    if vrange == 0.6:
        pct = 0.0009   # 0.09 %
        digits = 2
        res = 1e-4
    elif vrange == 6:
        pct = 0.0009
```

```
        digits = 2
        res = 1e-3
    elif vrange == 60:
        pct = 0.0009
        digits = 2
        res = 1e-2
    elif vrange == 600:
        pct = 0.0009
        digits = 2
        res = 1e-1
    elif vrange == 1000:
        pct = 0.0015   # 0.15 %
        digits = 2
        res = 1.0
    else:
        raise ValueError("Unbekannter Fluke-179-Spannungsbereich (vrange)")

    return pct * v + digits * res

def fluke179_dc_current_error(value, irange):
    """
    Geräteunsicherheit für Gleichstrom mit Fluke 179.
    'value'  in Ampere (Mittelwert),
    'irange' gewählter Messbereich: 0.06, 0.4, 6, 10 (A).

    Aus Leitfaden: für 60 mA, 400 mA, 6 A, 10 A:
        ±(1 % vom Messwert + 3 digits)
    Auflösungen:
        60 mA  : 0.01 mA  = 1e-5 A
        400 mA : 0.1 mA   = 1e-4 A
        6 A    : 0.001 A
        10 A   : 0.01 A
    """
    i = abs(value)

    if irange == 0.06:
        pct = 0.01
        digits = 3
        res = 1e-5
    elif irange == 0.4:
        pct = 0.01
        digits = 3
        res = 1e-4
    elif irange == 6:
        pct = 0.01
        digits = 3
        res = 1e-3
```

```
    elif irange == 10:
        pct = 0.01
        digits = 3
        res = 1e-2
    else:
        raise ValueError("Unbekannter Fluke-179-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 besserer Unsicherheit (Leitfaden + Fluke 179)
# ============================================================

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 Fluke-179-Spannungsbereich (z.B. 6, 60, 600)
    I_range: verwendeter Fluke-179-Strombereich (z.B. 0.4, 6, 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)

    # --- Gerätefehler des Fluke 179 (Typ-B-Unsicherheit)
    dU_gerät = fluke179_dc_voltage_error(U_mean, U_range)
    dI_gerät = fluke179_dc_current_error(I_mean, I_range)

    # --- Leitfaden-Regel: Delta = max(ux, Delta_gerät)
```

```
    dU = max(abs(u_U), abs(dU_gerät))
    dI = max(abs(u_I), abs(dI_gerät))

    # --- 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
```

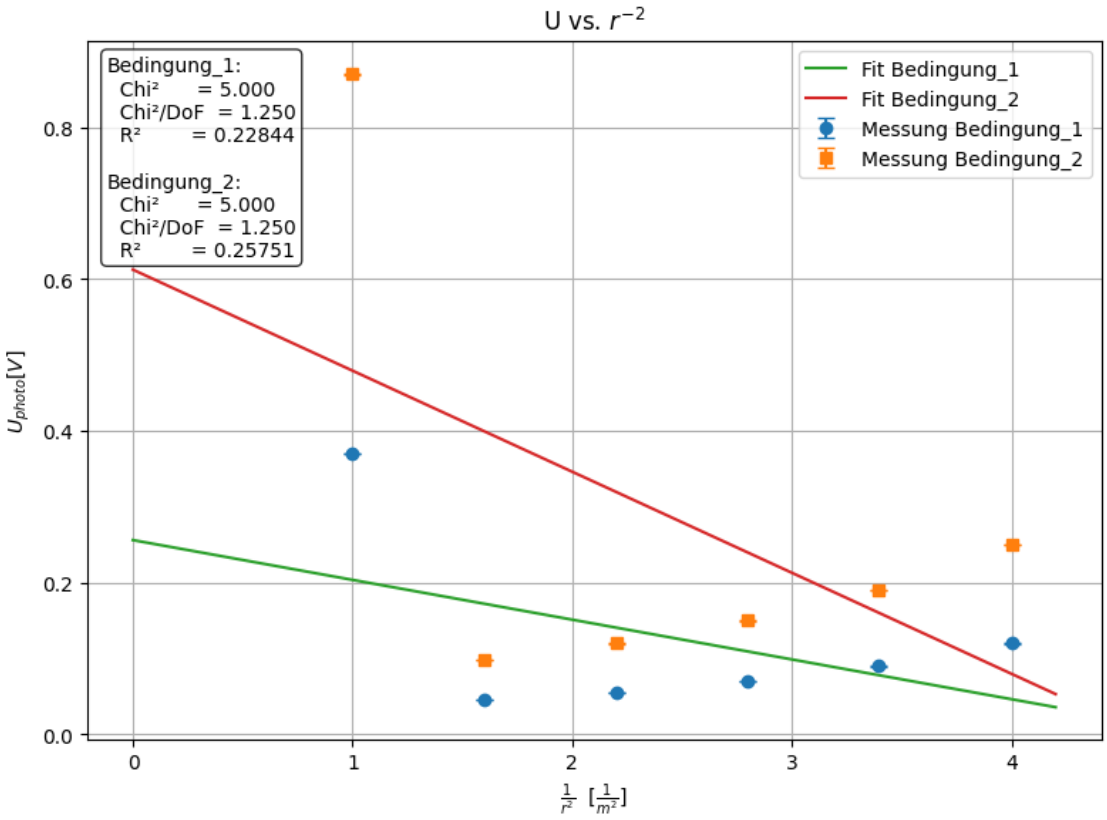
```
            def f(x): return compute_temperatures(I_ratio, P1, x)
        else:
            raise ValueError

        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
# ============================================================

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

    # -------- Unsicherheiten der Photospannung aus Fluke 179 --------
    # Angenommener Messbereich für U_photo (anpassen falls nötig):
    # z.B. 0.6 V oder 6 V, je nachdem, was ihr am DMM hattet.
    Uphoto_range = 0.6

    U1_err = np.array($$fluke179_dc_voltage_error(u, Uphoto_range) for u in␣
 ↪U1$$)
```

```
    U2_err = np.array($$fluke179_dc_voltage_error(u, Uphoto_range) for u in␣
↪U2$$)

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

    # Fehlerbalken mit Wert-abhängigen Fehlern
    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_{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($$500, 543, 598, 674, 791, 1000$$, dtype=float)
    U1_photo = np.array($$0.120, 0.090, 0.070, 0.055, 0.045, 0.37$$,␣
↪dtype=float)

    # Lampenspannung/-strom Messreihe (kann auch nur 1 Wert sein)
    U1_lamp = np.full_like(r1_mm, 6.00, dtype=float)   # Beispiel: 6 V
    I1_lamp = np.full_like(r1_mm, 0.30, dtype=float)   # Beispiel: 0.30 A

    # Fluke-179-Bereiche für diese Messung
    U1_range = 60   # z.B. 60 V-Bereich
    I1_range = 0.4  # z.B. 400 mA-Bereich

    label1 = "Bedingung_1"

    # --------- Bedingung 2 ----------
    r2_mm = np.array($$500, 543, 598, 674, 791, 1000$$, dtype=float)
    U2_photo = np.array($$0.250, 0.190, 0.150, 0.120, 0.098, 0.87$$,␣
↪dtype=float)

    U2_lamp = np.full_like(r2_mm, 12.00, dtype=float)  # Beispiel: 12 V
    I2_lamp = np.full_like(r2_mm, 0.45, dtype=float)   # Beispiel: 0.45 A

    U2_range = 60   # z.B. 60 V-Bereich
    I2_range = 0.4  # z.B. 400 mA-Bereich

    label2 = "Bedingung_2"

    # ========= 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 Fluke-179-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 Bedingung_1: U = a * (1/r²) + b
  a = (-5.241921e-02 ± 4.816829e-02) V m²
  b = (2.559390e-01 ± 1.300286e-01) V

Fit Bedingung_2: U = a * (1/r²) + b
  a = (-1.331517e-01 ± 1.130487e-01) V m²
  b = (6.122690e-01 ± 3.051711e-01) V

I1/I2 = 0.393680 ± 0.492529

Leistung Bedingung_1:
  U_mean = (6.0000 V ± 0.0254) V
  I_mean = (0.3000 A ± 0.0033) A
  P      = (1.8000 W ± 0.0212) W

Leistung Bedingung_2:
  U_mean = (12.0000 V ± 0.0308) V
  I_mean = (0.4500 A ± 0.0048) A
  P      = (5.4000 W ± 0.0592) W

Strahlungstemperaturen:
  T1 (Bedingung_1) = 6390.83 ± nan K
  T2 (Bedingung_2) = 8410.81 ± nan K


/tmp/ipykernel_61381/3602739720.py:213: RuntimeWarning: invalid value
encountered in log
  denominator = np.log(R_I)
```



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 ein 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:

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

3.7 Thermischer Widerstand

Der thermische Widerstand lautet:

$$R_{th} = \frac{\Delta T}{\dot{Q}}.$$

Fuer einen homogenen Stab gilt:

$$R_{th} = \frac{L}{\lambda A}.$$

3.8 Wärmeverluste

Reale Stabe verlieren Wärme an die Umgebung durch Konvektion und Strahlung:

$$\dot{Q} \cdot ver = hA \cdot Oberflche(T - T_{umgebung}).$$

Dies fuehrt zu leichten Abweichungen vom ideal linearen Temperaturverlauf und sollte im Protokoll kurz diskutiert werden.

[30]:
```
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]
        else:
            print(f"Spalte '{key}' nicht gefunden -> benutze Default.")
            result[key] = default_dict[key]

    return result


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

def fluke179_dc_voltage_error(value, vrange):
    v = abs(value)
    if vrange == 0.6:
        return 0.0009 * v + 2 * 1e-4
    elif vrange == 6:
        return 0.0009 * v + 2 * 1e-3
```

```
    elif vrange == 60:
        return 0.0009 * v + 2 * 1e-2
    elif vrange == 600:
        return 0.0009 * v + 2 * 1e-1
    elif vrange == 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, label=""):
    x_m = x_mm / 1000
    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.errorbar(x_m, T, yerr=T_err, fmt="o", capsize=4, label="Messpunkte")
    x_all = np.linspace(0, np.max(x_m)*1.05, 300)
    plt.plot(x_all, fit_func(x_all), "-", label="Fit")

    txt  = f"Chi²/DoF = {chi2_dof:.3f}\nR² = {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    = data["U_heiz"]
    I    = data["I_heiz"]

    # ------------------------------------------------------------
    # Geometrie: duennes rechteckiges Band
    # ------------------------------------------------------------
    # Beispielwerte: Breite b und Dicke h in m
    # HIER deine realen Werte und Unsicherheiten eintragen:
    b  = 0.020   # 20 mm Breite
    db = 0.0001  # 0.1 mm Unsicherheit
    h  = 0.002   # 2 mm Dicke
    dh = 0.0001  # 0.1 mm Unsicherheit

    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. Thermoelement ±0.5 K)
    T_err = np.full_like(T_C, 0.5)

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

    # ------------------------------------------------------------
    # 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(\"experiment2.csv\")
    main(None)
```

```
Keine CSV angegeben -> benutze eingebaute Daten.

Breite b = (0.0200 ± 0.0001) m
Dicke  h = (0.0020 ± 0.0001) m
Querschnittsflaeche A = (4.0000e-05 ± 2.0100e-06) m^2

Fit : T(x) = m*x + b
  m = (-2.1314e+02 ± 2.4702e+00) K/m
  b = (6.4320e+01 ± 1.9240e-01) K
```

Leistung: P = (11.424 ± 0.153) W

Lambda = (1339.91 ± 71.39) W/(mK)