Licht auf Abwegen: Wie Naturgesetze die Richtung bestimmen

Mathis Fricke (fricke@mma.tu-darmstadt.de)

20.01.2025

Einleitung

Die Natur zeigt oft eine bemerkenswerte Eigenschaft: Sie wählt von sich aus effiziente Lösungen. Viele Naturgesetze lassen sich durch Prinzipien beschreiben, bei denen bestimmte Größen minimiert oder maximiert werden. Dies ist bekannt als das Prinzip der kleinsten Wirkung oder allgemein als Variationsprinzipien.

Ein klassisches Beispiel ist die Lichtbrechung: Wenn ein Lichtstrahl von einem Medium in ein anderes übergeht (z. B. von Luft in Wasser), wählt das Licht den Weg, bei dem die gesamte Laufzeit minimiert wird. Dieses Phänomen kann mit Hilfe der Variationsrechnung erklärt werden und führt direkt auf das Snell'sche Gesetz.

Die folgende Abbildung zeigt das Phänomen der Lichtbrechung an der Grenzfläche zwischen zwei Medien:

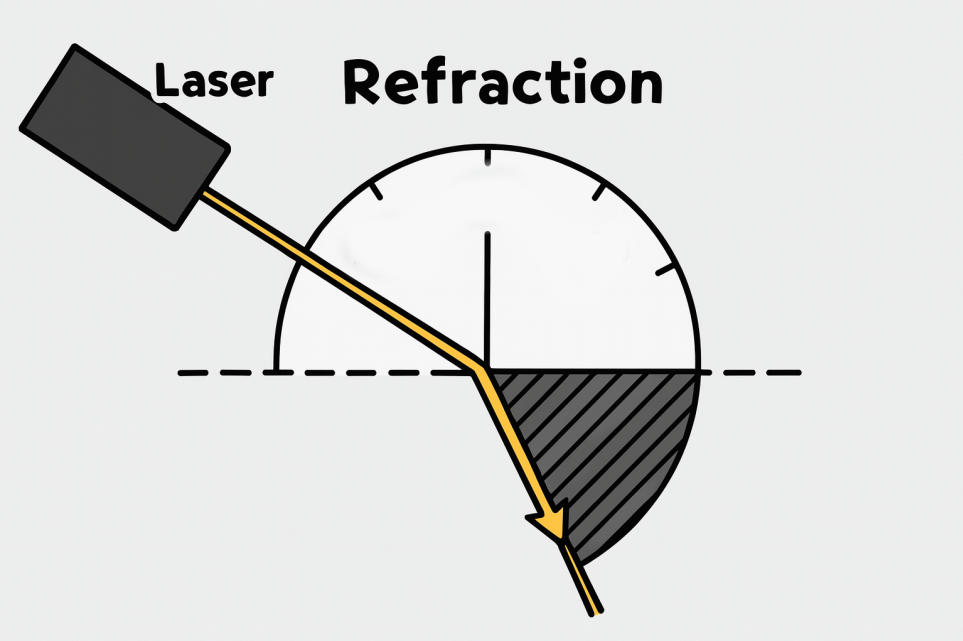

Figure 1: Lichtbrechung an der Grenzfläche zwischen zwei Medien.

In diesem Handout werden wir Schritt für Schritt untersuchen, wie die Variationsrechnung genutzt werden kann, um die Lichtbrechung zu beschreiben, und wie sich daraus das Snell'sche Gesetz ergibt. Als Einstieg, siehe auch das Video
`https://youtu.be/EAK1FEGs8ZI?si=GCsivTnAtym1eEI9`.

Mathematische Modellierung

Gegeben seien:

- Startpunkt des Lichtstrahls: $(0, 0)$.
- Zielpunkt des Lichtstrahls: (x_z, y_z).
- Eine Grenzfläche zwischen zwei Medien mit Brechungsindizes n_1 und n_2 bei $x = D$.
- Der Schnittpunkt des Lichtstrahls mit der Grenze: (D, y^*).

Die Gesamtlaufzeit des Lichtstrahls ergibt sich als Summe der Laufzeiten in den beiden Medien:

$$T(y^*) = n_1 L_1 + n_2 L_2 = n_1 \sqrt{D^2 + y^{*2}} + n_2 \sqrt{(x_z - D)^2 + (y_z - y^*)^2}. \tag{1}$$

Das Ziel ist es, y^* so zu wählen, dass $T(y^*)$ minimiert wird.

Numerische Visualisierung mit Python

Der folgende Python-Code berechnet die Gesamtlaufzeit $T(y^*)$ für verschiedene Werte von y^*, bestimmt das Optimum numerisch und visualisiert den Strahlengang für das optimale y^*:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

def gesamte_laufzeit(y_stern, n1, n2, D, x_z, y_z):
    """
    Berechnet die gesamte Laufzeit T in Abhaengigkeit von y*.

    Parameter:
    y_stern (float): Schnittpunkt auf der Grenze bei x = D.
    n1 (float): Brechungsindex des Mediums 1.
    n2 (float): Brechungsindex des Mediums 2.
    D (float): x-Koordinate der Grenze zwischen den beiden Medien.
    x_z (float): x-Koordinate des Zielpunkts.
    y_z (float): y-Koordinate des Zielpunkts.

    Rueckgabe:
    float: Gesamte Laufzeit.
    """
    L1 = np.sqrt(D**2 + y_stern**2)  # Weglaenge im Medium 1
    L2 = np.sqrt((x_z - D)**2 + (y_z - y_stern)**2)  # Weglaenge im Medium 2
    return n1 * L1 + n2 * L2

# Parameter
n1 = 1.0  # Brechungsindex des Mediums 1
```

```
n2 = 1.5  # Brechungsindex des Mediums 2
D = 3.0   # x-Koordinate der Grenze
x_z = 6.0 # x-Koordinate des Zielpunkts
y_z = 4.0 # y-Koordinate des Zielpunkts

# Bereich der y*-Werte
y_stern_werte = np.linspace(0, y_z, 500)

# Berechnung der Laufzeiten
laufzeiten = [gesamte_laufzeit(y_stern, n1, n2, D, x_z, y_z) for
    y_stern in y_stern_werte]

# Numerische Optimierung, um das Minimum von T(y*) zu finden
result = minimize_scalar(lambda y_stern: gesamte_laufzeit(y_stern
    , n1, n2, D, x_z, y_z), bounds=(0, y_z), method='bounded')
optimal_y_stern = result.x

# Plot der Gesamtlaufzeit
plt.figure(figsize=(8, 5))
plt.plot(y_stern_werte, laufzeiten, label='Gesamte Laufzeit',
    color='blue')
plt.axvline(x=optimal_y_stern, color='red', linestyle='--', label
    =f'Optimum: y* = {optimal_y_stern:.2f}')
plt.title("Gesamte Laufzeit in Abhaengigkeit vom Schnittpunkt y*"
    )
plt.xlabel("Schnittpunkt y* (bei x = D)")
plt.ylabel("Gesamte Laufzeit")
plt.legend()
plt.grid()
plt.savefig("gesamte_laufzeit.pdf")
plt.close()

# Visualisierung des Strahlengangs
plt.figure(figsize=(8, 5))
plt.plot([0, D, x_z], [0, optimal_y_stern, y_z], label='
    Strahlengang', marker='o', color='blue')
plt.plot([0, x_z], [0, y_z], linestyle='--', label='
    Direktverbindung', color='gray')
plt.title("Visualisierung des Strahlengangs")
plt.xlabel("x")
plt.ylabel("y")
plt.axvline(x=D, color='black', linestyle='--', label='Grenze bei
     x = D')
plt.text(D / 2, optimal_y_stern / 2, f'n1 = {n1}', fontsize=10,
    color='blue')
plt.text((D + x_z) / 2, (optimal_y_stern + y_z) / 2, f'n2 = {n2}'
    , fontsize=10, color='blue')
plt.legend()
plt.grid()
plt.savefig("strahlengang.pdf")
plt.close()
```

Listing 1: Python-Code zur Berechnung und Visualisierung

Einbindung der Abbildungen

Die Abbildungen, die mit dem Python-Code erzeugt wurden, sind nachfolgend dargestellt:

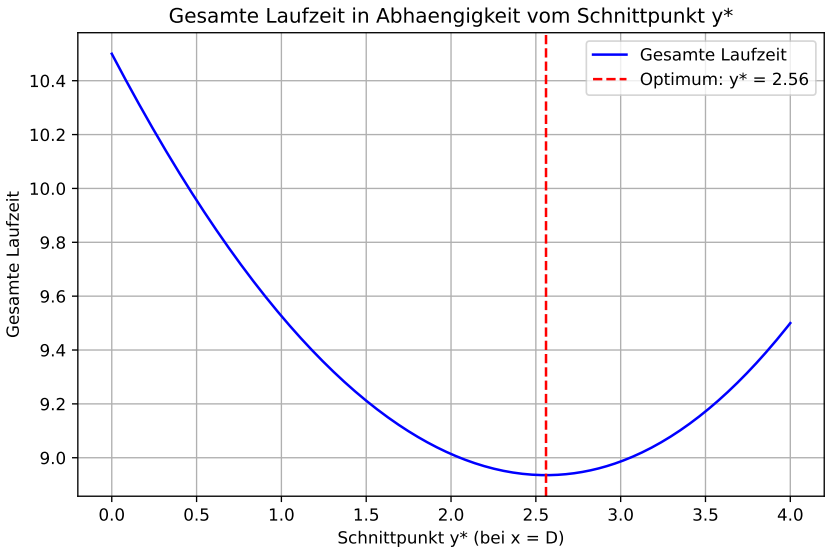

Figure 2: Gesamte Laufzeit in Abhaengigkeit vom Schnittpunkt y^*.

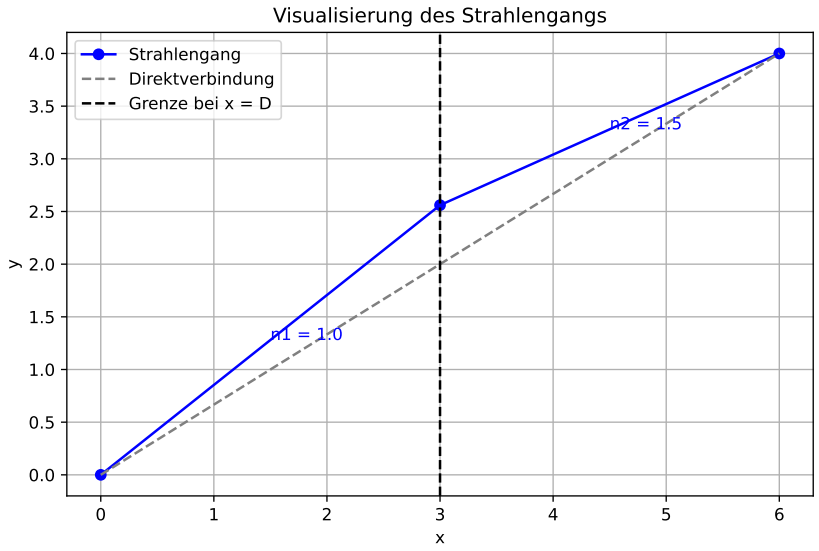

Figure 3: Visualisierung des Strahlengangs bei optimalem y^*.

Zusammenhang zum Snell'schen Gesetz

Die Minimierung der Gesamtlaufzeit $T(y^*)$ führt auf die Bedingung, dass die Ableitung von $T(y^*)$ nach y^* verschwindet:

$$\frac{dT}{dy^*} = n_1 \frac{y^*}{\sqrt{D^2 + y^{*2}}} - n_2 \frac{y_z - y^*}{\sqrt{(x_z - D)^2 + (y_z - y^*)^2}} = 0. \tag{2}$$

Diese Gleichung beschreibt das Gleichgewicht der Steigungen (Winkel), die das Licht im ersten und zweiten Medium einnimmt. Durch Umformungen zeigt sich, dass dies genau dem Snell'schen Gesetz entspricht:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2, \tag{3}$$

wobei:

$$\sin\theta_1 = \frac{y^*}{\sqrt{D^2 + y^{*2}}},$$
$$\sin\theta_2 = \frac{y_z - y^*}{\sqrt{(x_z - D)^2 + (y_z - y^*)^2}}.$$

Das Snell'sche Gesetz beschreibt die Brechung von Licht an einer Grenzfläche zwischen zwei Medien. Die Winkel θ_1 und θ_2 beziehen sich dabei auf den Einfalls- bzw. Brechungswinkel des Lichtstrahls.

Zusammenfassung

In diesem Handout wurde gezeigt, wie man mit der Variationsrechnung das Minimum der Laufzeit eines Lichtstrahls durch zwei Medien bestimmen kann. Der Python-Code bietet eine numerische Visualisierung dieses Problems und illustriert das Minimum anschaulich. Dies führt direkt auf das Snell'sche Gesetz der Lichtbrechung:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2. \tag{4}$$

Der Python Code und diese Zusammenfassung sind hier verfügbar:
`https://github.com/MathisFricke/Licht-auf-Abwegen`.