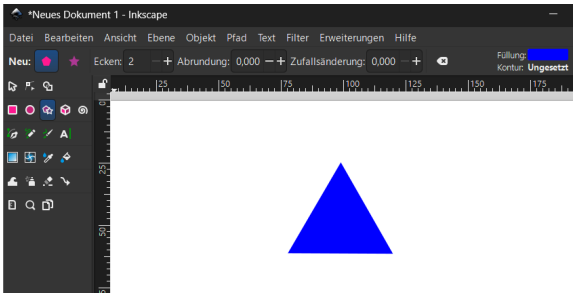
Blatt 11

Vincent Kümmerle und Elvis Gnaglo

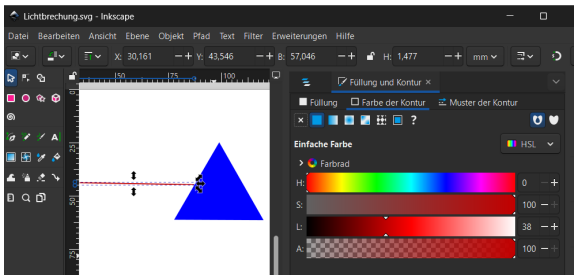
10. Januar 2026

1 Erstellen einer Vektorgrafik

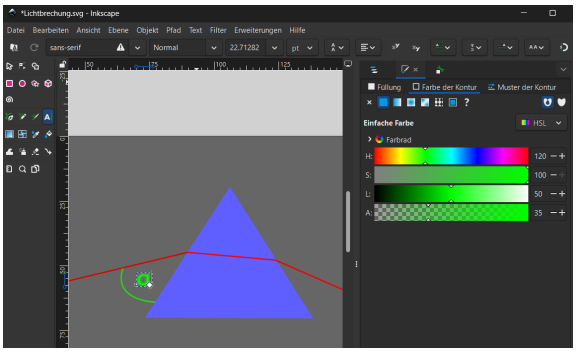
Die Vektorgrafik zur Brechung eines Laserstrahls an einem Prisma wurde mit Inkscape erstellt. Die einzelnen Schritte sind in den folgenden Abbildungen dargestellt.



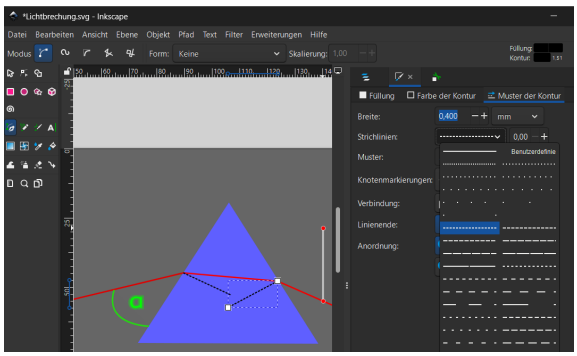
(a) Erstellung des Dreiecks mit Polygon-Werkzeug.



(b) Erstellung des Laserstrahls.



(a) Hinzufügen der gebrochenen Strahlen mit dem Bezier-Werkzeug und des Winkels als Text.



(b) Erstellung des Einfalls- und Ausfallslots.

Im letzten Schritt wurde die Seitengröße an den Rahmen um alle Objekte angepasst und die Grafik als PDF exportiert.

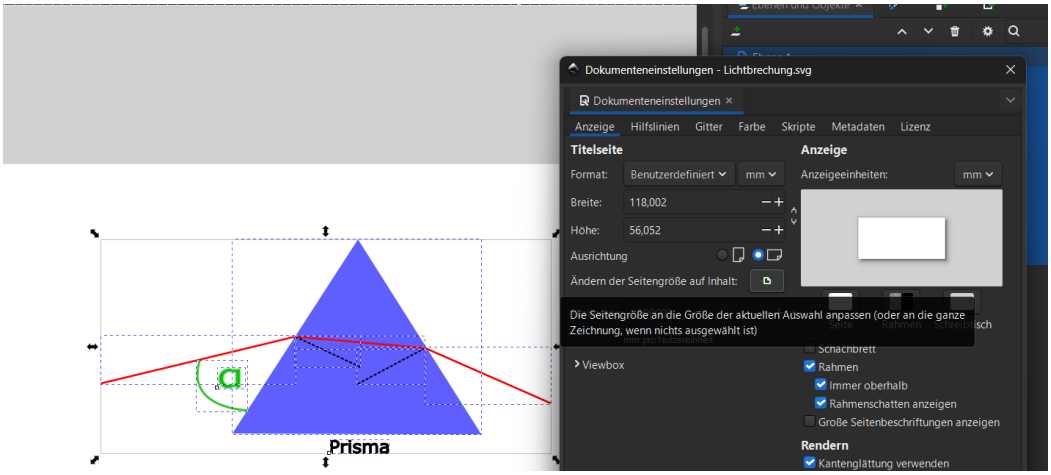


Abbildung 3: Anpassung der Seitengröße an den Rahmen um alle Objekte.

Die erstellte Vektorgrafik ist in Abbildung 4 dargestellt. Sie zeigt die Brechung eines monochromatischen Laserstrahls an einem Prisma mit dem Einfallswinkel α, dem Einfalls- und Ausfallslot sowie den gebrochenen Strahlen im Prisma.

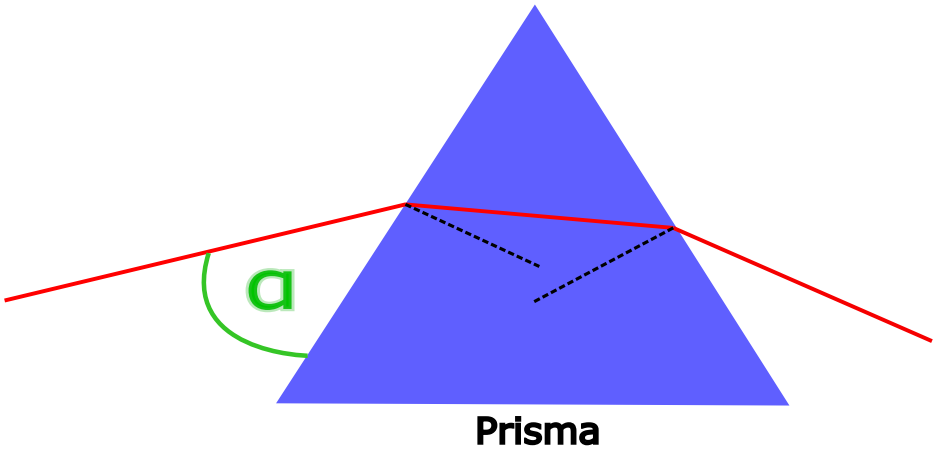


Abbildung 4: Strahlengang eines monochromatischen Laserstrahls an einem Prisma mit Einfallswinkel α.

2 Lineare Algebra mit NumPy: Fibonacci-Folge

Das vorliegende Python-Skript untersucht das Konvergenzverhalten des Verhältnisses aufeinanderfolgender Fibonacci-Zahlen a_n/a_{n-1} mithilfe von Matrix-Vektor-Multiplikationen und einer Eigenwertanalyse.
Zunächst werden die Anfangswerte der Fibonacci-Folge als Vektor $v = (1, 1)^T$ initialisiert, was den Startwerten $a_2 = 1$ und $a_1 = 1$ entspricht. Die Iterationsvorschrift wird durch die Matrix M definiert:

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \tag{1}$$

Die resultierenden Verhältnisse werden graphisch als diskrete Punkte über der Anzahl der Iterationen aufgetragen.
Im zweiten Teil des Skripts wird das Verhalten der Folge durch die spektralen Eigenschaften der Matrix M erklärt. Mithilfe der Funktion `np.linalg.eig` werden die Eigenwerte und Eigenvektoren von M numerisch bestimmt.
Da M symmetrisch ist, existieren reelle Eigenwerte. Diese werden nach ihrem Betrag sortiert, um den dominanten Eigenwert λ_1 zu identifizieren. Theoretisch ergeben sich die Eigenwerte aus dem charakteristischen Polynom $\lambda^2 - \lambda - 1 = 0$ zu:

$$\lambda_{1,2} = \frac{1 \pm \sqrt{5}}{2} \tag{2}$$

Der dominante Eigenwert ist der Goldene Schnitt $\lambda_1 \approx 1.618$.
Das Skript vergleicht die iterative Lösung mit diesem theoretischen Grenzwert. Dazu wird eine konstante Linie (rote gestrichelte Linie) auf Höhe von λ_1 in den Plot eingezeichnet.
Die graphische Auswertung zeigt, dass das Verhältnis a_n/a_{n-1} sehr schnell gegen den dominanten Eigenwert λ_1 konvergiert. Anfangs sind Oszillationen sichtbar, die jedoch rasch abklingen, da der Einfluss des betragsmäßig kleineren Eigenwertes $|\lambda_2| < 1$ mit $n \to \infty$ exponentiell verschwindet.

```
import numpy as np
import matplotlib.pyplot as plt

# Part 1
v = np.array([1.0, 1.0])
M = np.array([[1, 1],
              [1, 0]])

ratios = []
n_values = range(50)

for i in range(50):
    current_ratio = v[0] / v[1]
    ratios.append(current_ratio)
    v = M @ v

plt.figure(figsize=(10, 6))
plt.plot(n_values, ratios, linestyle="none", marker="o", label="
    Numerisches Verhältnis $a_n/a_{n-1}$", markersize=4)


# Part 2
# Berechnung der Eigenwerte und Eigenvektoren
```

```
eigenvalues, eigenvectors = np.linalg.eig(M)

# Sortierung nach dem größten Eigenwert
idx = np.argsort(np.abs(eigenvalues))[::-1]
lambdas = eigenvalues[idx]

print(f"Die Eigenwerte von M sind: {lambdas}")
print(f"Dominanter Eigenwert (lambda_1): {lambdas[0]}")

# Approximation:
approx_ratios = [lambdas[0]] * len(n_values)

# Plotten der Approximation
plt.plot(n_values, approx_ratios, color='red', linestyle='--', label=
    f"Approximation ($\\lambda_1 \\approx {lambdas[0]:.4f}$)")


plt.title("Konvergenz des Verhältnisses von Fibonacci-Zahlen")
plt.xlabel("Iteration n")
plt.ylabel("Verhältnis $a_n / a_{n-1}$")
plt.legend()
plt.grid(True, which="both", linestyle='--', alpha=0.7)
plt.savefig('fibonacci.pdf')
plt.show()
```

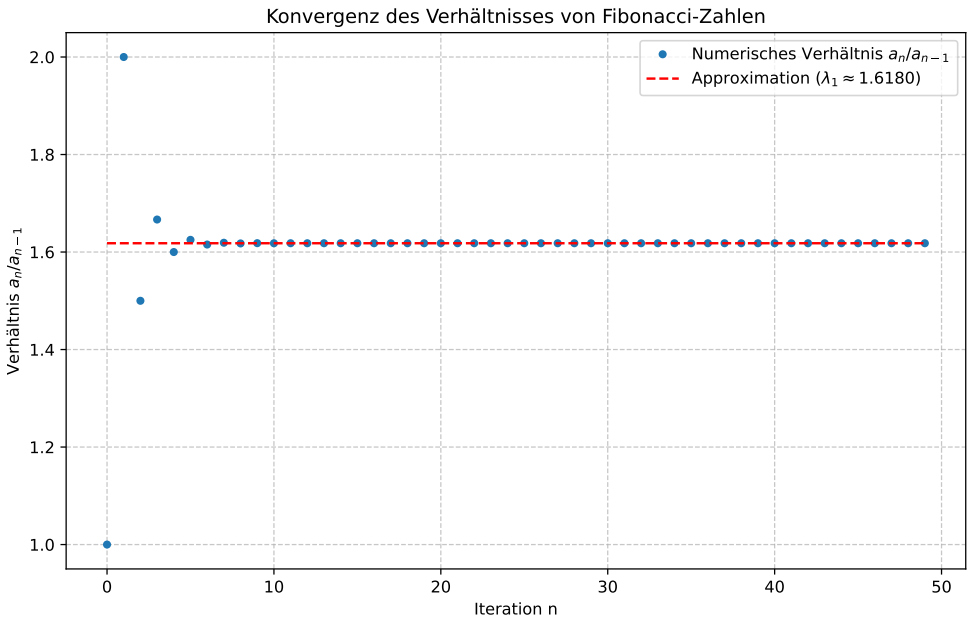


Abbildung 5: Die geplotteten Verhältnisse der Fibonacci Folge mit der berechneten Approximation.

3 Quicksort-Algorythmus

```
    def quick_sort(arr):
    # Basisfall: Die Liste ist leer ist oder hat nur ein Element
    if len(arr) <= 1:
        return arr

    # 1. Wahl des Pivot-Elements (mittleres Element)
    pivot = arr[len(arr) // 2]

    # 2. Aufteilen der Liste in drei Teile
    # Elemente < Pivot
    left = [x for x in arr if x < pivot]

    # Elemente = Pivot (Duplikate)
    middle = [x for x in arr if x == pivot]

    # Elemente > Pivot
    right = [x for x in arr if x > pivot]

    # 3. Rekursiver Aufruf und Zusammenfügen der Ergebnisse
    return quick_sort(left) + middle + quick_sort(right)

#Beispiel
b_liste = [3, 6, 8, 10, 1, 2, 1, 90, 4, 9, 0, -1]
s_liste = quick_sort(b_liste)

print(f"Original: {b_liste}")
print(f"Sortiert: {s_liste}")
```