import numpy as np  
from scipy.optimize import minimize  
  
# Function to calculate the intersection point between two lines  
def intersection\_point(line1, line2):  
 x1, y1, theta1 = line1  
 x2, y2, theta2 = line2  
 A = np.array([[np.cos(theta1), -np.cos(theta2)], [np.sin(theta1), -np.sin(theta2)]])  
 b = np.array([x2\*np.sin(theta2)-y2\*np.cos(theta2), x1\*np.sin(theta1)-y1\*np.cos(theta1)])  
 intersection = np.linalg.solve(A, b)  
 return intersection  
  
# Function to calculate the reflection angle  
def reflection\_angle(incident\_angle, surface\_angle):  
 return 2 \* surface\_angle - incident\_angle  
  
# Function to calculate the position and angle of each straight line segment  
def calculate\_segments(num\_segments, length\_ab):  
 segment\_length = length\_ab / num\_segments  
 segment\_positions = np.linspace(0, length\_ab, num\_segments+1)[:-1]  
 segment\_angles = np.pi / 2 \* np.ones(num\_segments)  
 return segment\_positions, segment\_angles  
  
# Function to calculate the proportion of light entering the CD interval after two reflections  
def calculate\_proportion\_cd(segment\_positions, segment\_angles, length\_cd):  
 num\_segments = len(segment\_positions)  
 incident\_angle = np.pi / 2  
 intersection\_points = [(0, 0)]  
  
 for i in range(num\_segments):  
 # Calculate the intersection point between incident ray and each segment  
 intersection = intersection\_point(intersection\_points[-1] + [0, segment\_positions[i], incident\_angle],  
 [segment\_positions[i], 400, segment\_angles[i]])  
 intersection\_points.append(intersection)  
  
 # Calculate the reflection angle  
 incident\_angle = reflection\_angle(incident\_angle, segment\_angles[i])  
  
 # Calculate the intersection point between last reflected ray and CD  
 intersection = intersection\_point(intersection\_points[-1] + [0, segment\_positions[-1], incident\_angle],  
 [segment\_positions[-1], 400, 0])  
 intersection\_points.append(intersection)  
  
 # Count the number of points within CD interval  
 count\_cd = sum(0 <= point[0] <= length\_cd for point in intersection\_points)  
 proportion\_cd = count\_cd / (num\_segments + 1)  
  
 return proportion\_cd, intersection\_points  
  
# Objective function to maximize the proportion of light entering CD interval  
def objective\_function(x):  
 segment\_positions, segment\_angles = calculate\_segments(int(x), 400)  
 proportion\_cd, \_ = calculate\_proportion\_cd(segment\_positions, segment\_angles, 10)  
 return -proportion\_cd, segment\_positions, segment\_angles  
  
# Initial guess for the number of segments  
x0 = np.array([5])  
  
# Bounds for the number of segments  
bounds = [(1, 20)]  
  
# Optimization  
result = minimize(lambda x: objective\_function(x)[0], x0, bounds=bounds)  
  
# Get optimal number of segments and corresponding EF information  
optimal\_segments = int(result.x)  
segment\_positions, segment\_angles = objective\_function(result.x)  
  
# Print the optimal number of segments  
print("Optimal number of segments:", optimal\_segments)  
  
# Print specific information about EF  
print("Lengths of segments:", segment\_positions)  
print("Angles of segments:", segment\_angles)