Санкт-Петербургский Государственный Электротехнический

Университет "ЛЭТИ"

кафедра физики

Задание №1 по дисциплине

"Физические основы информационных технологий"

Название: Искривление луча в оптическом канале

|  |  |
| --- | --- |
| Фамилия И.О.: | Бутыло Е. А. |
| группа: | 1303 |
| Преподаватель: | Альтмарк А. М. |
| Итоговый балл: |  |
| Крайний срок сдачи: | 22.10.23 |

Санкт-Петербург 2023

Условие задания

Найти длину траектории светового луча *S* в прямолинейном дисперсионном оптоволоконном канале, Рис.1, c показателем преломления n1. Оптоволокно окружено средой с показателем преломления n2. Функцию распределения показателя преломления n1(y, ω) можно представить как:

,

где y – поперечная координата, ω – циклическая частота светового луча.

Функцию , функцию Zf(y), описывающую координату z выходного торца волновода, начальный угол ввода луча α в волновод, координату ввода луча в волновод *y*=y0, радиус канала R можно взять в файле FOIT\_IDZ1.xlsx. Все геометрические размеры даются в безразмерных координатах.

Необходимо построить график траектории луча, а также записать ответ *S* в текстовый файл IDZ1\IDZ1.txt. Помимо текстового файла IDZ1.txt в папке IDZ1 должен находиться Word-файл (Pdf-файл) с отчетом, а также файл с кодом (Python, Mathcad, Mathematica).

n2

Y

R

Zf(y)

y0

n1

Z

α

0

n2

-R

n2

Рисунок.1

**Таблица с исходными данными**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вар | R | n2 | f1(y) | Zf(y) | ω\*10^14, рад/с | y0 | α, град |
| 3 | 0.4 | 1 | 1.8 - 0.15\*y^2 | 50 + 3\*sin[17.951958020513104\*y] | 3.2 | 0.2 | 10 |

**Основные теоретические положения**

**Закон преломления**

Если световые волны достигают границы раздела двух сред и проникают в другую среду, то направление их распространения также изменяется — происходит преломление света. Преломление света — это изменение направления распространения световой волны при переходе из одной среды в другую.

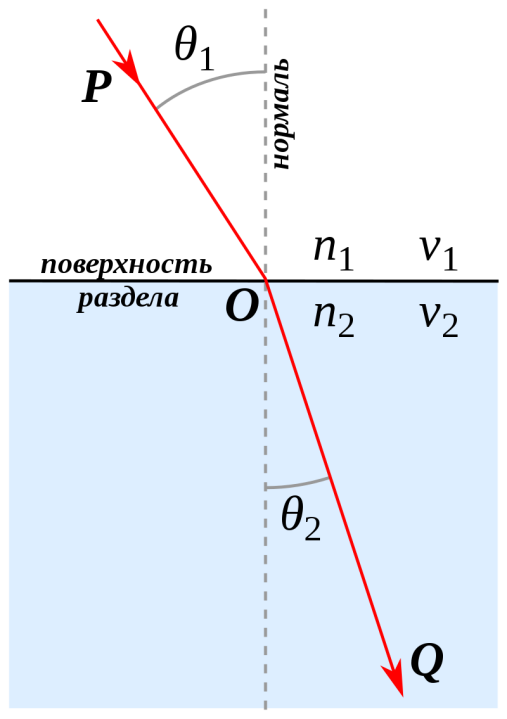
Угол падения света на поверхность связан с углом преломления соотношением:

где n1 — показатель преломления среды, из которой свет падает на границу раздела;

— угол падения света — угол между падающим на поверхность лучом и нормалью к поверхности;

— показатель преломления среды, в которую свет попадает, пройдя границу раздела;

— угол преломления света — угол между прошедшим через поверхность лучом и нормалью к поверхности.



Если , имеет место полное внутреннее отражение (преломлённый луч отсутствует, падающий луч полностью отражается от границы раздела сред).

**Оптическое волокно** — нить из оптически прозрачного материала (стекло, пластик), используемая для переноса света внутри себя посредством полного внутреннего отражения.

**Полное внутреннее отражение** — внутреннее отражение, при условии, что угол падения превосходит некоторый критический угол. Учитывая, что угол преломления не может превышать 90°, получаем, что при угле падения, синус которого больше отношения меньшего показателя преломления к большему показателю, электромагнитная волна должна полностью отражаться в первую среду.

**Вывод**

С помощью Python была рассчитана траектория хода луча в прямолинейном дисперсионном оптоволоконном канале с показателем

преломления, задаваемым формулой. Полученная траектория была построена на графике, а также была рассчитана её длина.

**ПРИЛОЖЕНИЕ А**

**ПРОГРАММА MAIN.PY**

import matplotlib.pyplot as plt

from math import \*

def Zf(y) -> float:

return 50 + 3 \* sin(17.951958020513104 \* y)

def f1(y: float) -> float:

return 1.8 - 0.15 \* pow(y, 2)

def coefficient\_n1(w: float, ) -> float:

return 1 - pow(0.35 / w, 2)

def get\_start\_step(y0: float, steps: dict) -> int or None:

idx = 0

while idx < len(steps) / 2:

if steps[idx][0] < y0 < steps[idx][1]:

return idx

idx += 1

def get\_n1(y: float) -> float:

return f1(y) \* coefficient

def main() -> None:

plt.figure(figsize=(15, 5))

plt.xlabel("Z", loc="right")

plt.ylabel("Y", loc="top")

plt.locator\_params(axis='x', nbins=20)

plt.ylim(-radius, radius)

plt.xlim(0, 55)

# stats for layers

cf\_on\_step = {i - ((N - 1) / 2): get\_n1(i \* step + step / 2 - radius) for i in range(N)}

z\_exit = {i - ((N - 1) / 2): Zf(i \* step + step / 2 - radius) for i in range(N)}

steps = {i - ((N - 1) / 2): (i \* step - radius, (i + 1) \* step - radius) for i in range(N)}

for key, value in z\_exit.items():

plt.plot(value, key \* step + step / 2,

marker="o", markersize=0.5, markeredgecolor='black', markerfacecolor="black")

# start values

n\_alpha = n2

n\_beta\_ind = get\_start\_step(start\_y, steps)

alpha = radians(start\_alpha)

flag = 0

z = 0

length = 0

# first step

beta = asin(sin(alpha) \* n\_alpha / cf\_on\_step[n\_beta\_ind])

plt.plot([z, z + (steps[n\_beta\_ind][1] - start\_y) / tan(beta)], [start\_y, steps[n\_beta\_ind][1]])

length += (steps[n\_beta\_ind][1] - start\_y) / sin(beta)

z += (steps[n\_beta\_ind][1] - start\_y) / tan(beta)

alpha = pi / 2 - beta

n\_alpha = cf\_on\_step[n\_beta\_ind]

n\_alpha\_ind = n\_beta\_ind

n\_beta\_ind += 1

while True:

if z >= z\_exit[n\_alpha\_ind]:

# get delta z

z\_to\_edge = z\_exit[n\_alpha\_ind] - (z - step \* tan(alpha))

# get previous value of length and add length of trajectory to edge

length -= step / cos(alpha)

length += z\_to\_edge / sin(alpha)

break

if sin(alpha) > (cf\_on\_step[n\_beta\_ind] / n\_alpha):

# обработка случая отражения

if not flag:

plt.plot([z, z + step \* tan(alpha)], [steps[n\_beta\_ind][0], steps[n\_beta\_ind - 2][1]],

color='blue')

flag += 1

n\_beta\_ind -= 2

else:

plt.plot([z, z + step \* tan(alpha)], [steps[n\_beta\_ind][1], steps[n\_beta\_ind + 2][0]],

color='red')

flag -= 1

n\_beta\_ind += 2

else:

# обработка стандартного исхода

beta = asin(sin(alpha) \* n\_alpha / cf\_on\_step[n\_beta\_ind])

if flag:

plt.plot([z, z + step \* tan(beta)], [steps[n\_beta\_ind][1], steps[n\_beta\_ind][0]],

color='blue')

n\_alpha = cf\_on\_step[n\_beta\_ind]

n\_alpha\_ind = n\_beta\_ind

n\_beta\_ind -= 1

if not flag:

plt.plot([z, z + step \* tan(beta)], [steps[n\_beta\_ind][0], steps[n\_beta\_ind][1]],

color='red')

n\_alpha = cf\_on\_step[n\_beta\_ind]

n\_alpha\_ind = n\_beta\_ind

n\_beta\_ind += 1

alpha = beta

length += step / cos(alpha)

z += step \* tan(alpha)

print(f"length = {length}")

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# consts

radius: float = 0.4

n2: int = 1

start\_y: float = 0.2

start\_alpha: int = 10

# get n1 coefficient

omega: float = 3.2

coefficient: float = coefficient\_n1(omega)

# get steps

N: int = 100001

step: float = 2 \* radius / N

main()