Моделирование 2

Абросимов Арсений

June 2025

1 Задание “Моделирование интерференции”

Моделирование интерференции от N (1<=N<=10) узких высоких щелей с изменяемыми параметрами (ширина, период). Рассмотреть монохроматический и квазимонохроматический свет (задается середина и ширина спектра в нанометрах). Вывод цветного распределения интенсивности на выбранном расстоянии от щелей и графика зависимости интенсивности от координаты.

Для вычисления значения интерференции в точке x экрана для монохроматического света воспользуемся формулой:

$$I(x) = \left(\frac{\sin\beta}{\beta}\right)^2 \left(\frac{\sin(N\alpha)}{\sin\alpha}\right)^2 \tag{1}$$

Где $\left(\frac{\sin\beta}{\beta}\right)^2$ - дифракционный множитель(от 1 щели), а $\left(\frac{\sin(N\alpha)}{\sin\alpha}\right)^2$ - интерференционный множитель(от N щелей)

Параметры множителей вычисляются как:

$$\sin\theta = \frac{x}{\sqrt{x^2 + L^2}} \tag{2}$$

$$\alpha = \frac{\pi d \sin\theta}{\lambda} \tag{3}$$

$$\beta = \frac{\pi b \sin\theta}{\lambda} \tag{4}$$

Для примерного вычисления цвета интерференционной картины от длины волны $rgb(\lambda)$ воспользуемся интерполяцией по диапазонам длин волн(например 380-440 нм: фиолетовый и тд.), более полная таблица с коэффициентами есть в коде.

Если есть необходимость более точного вычисления можно воспользоваться библиотекой color-science

Для квазимонохроматического света диапазон с центром λ_0 и шириной $d\lambda$ будет выглядеть как $\left[\lambda_0 - \frac{d\lambda}{2}; \lambda_0 + \frac{d\lambda}{2}\right]$

Разобьем его на n частей $\lambda_i = \lambda_0 - \frac{d\lambda}{2} + i\frac{d\lambda}{n}, i \in [0; n]$ с некоторым шагом и возьмем вес W_i каждой части как $\frac{1}{n}$

Тогда интенсивность:

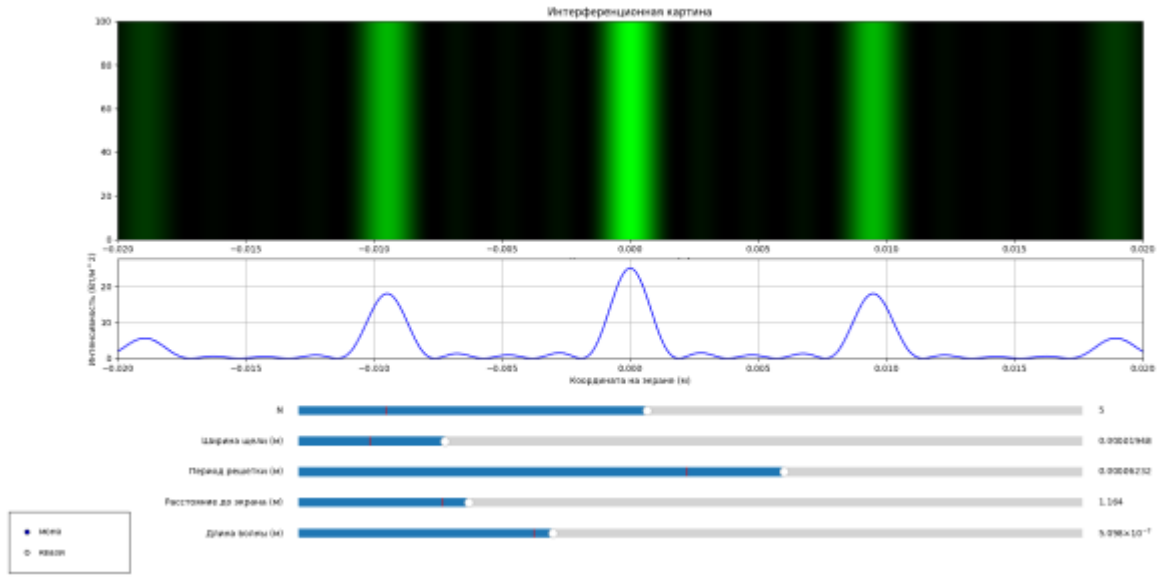
$$I(x) = \sum_{i=0}^{n} W_i I_{mono}(x, \lambda_i) \tag{5}$$

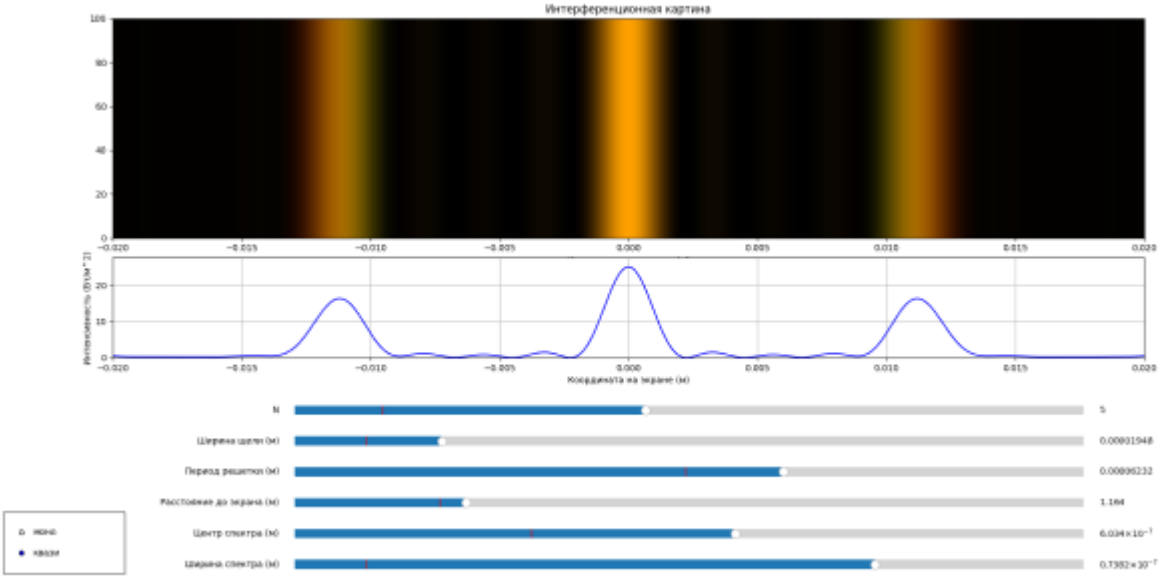
А цвета:

$$R(x) = \sum_{i=0}^{n} W_i I_{mono}(x, \lambda_i) r(\lambda_i) \tag{6}$$

Аналогично для $G(x)$ и $B(x)$

Примеры работы программы:





Код программы:

```
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, RadioButtons
from scipy.stats import norm

# Таблица примерных границ длин волн для разных цветов (нм)
wavelengths = np.array([380, 440, 490, 510, 580, 645, 750])
rgbs = np.array([
    [0.19, 0.00, 0.39],  # фиолетовый
    [0.00, 0.00, 1.00],  # синий
    [0.00, 1.00, 1.00],  # голубой
    [0.00, 1.00, 0.00],  # зелёный
    [1.00, 1.00, 0.00],  # жёлтый
    [1.00, 0.00, 0.00],  # красный
    [0.50, 0.00, 0.00]   # тёмно-красный
])

# Интерполяции для r, g, b компонент цвета
r_interp = interp1d(wavelengths, rgbs[:, 0], bounds_error=False, fill_value=0.0)
g_interp = interp1d(wavelengths, rgbs[:, 1], bounds_error=False, fill_value=0.0)
b_interp = interp1d(wavelengths, rgbs[:, 2], bounds_error=False, fill_value=0.0)

# Перевод длины волны в RGB
def wavelength_to_rgb(wavelength_nm):
    r = float(np.clip(r_interp(wavelength_nm), 0, 1))
    g = float(np.clip(g_interp(wavelength_nm), 0, 1))
    b = float(np.clip(b_interp(wavelength_nm), 0, 1))
    return r, g, b

# I(x) для монохроматической волны
def intensity_mono(x, N, b, d, L, lambda_):
    # Геометрически без приближений вычисляем sin(theta) = x/sqrt(x^2 + L^2)
    sin_theta = x / np.sqrt(x**2 + L**2)
    # alpha = pi*d*sin(theta)/lambda
    alpha = np.pi * d * sin_theta / lambda_
    # beta = pi*b*sin(theta)/lambda
    beta = np.pi * b * sin_theta / lambda_


    # I(x) = (sin(beta)/beta)^2 * (sin(N*alpha)/sin(alpha))^2
    with np.errstate(divide='ignore', invalid='ignore'):
        # (sin(beta)/beta)^2 с обработкой beta -> 0
        term1 = np.where(np.abs(beta) < 1e-10, 1.0, (np.sin(beta)/beta)**2)

        # (sin(N*alpha)/sin(alpha))^2 с обработкой alpha -> 2*pi*k, k in Z
        alpha_mod = np.mod(alpha, 2*np.pi)
        near_pi = np.abs(alpha_mod - np.pi) < 1e-10
        term2 = np.where(near_pi, N**2, (np.sin(N*alpha)/np.sin(alpha))**2)
        term2 = np.where(np.abs(alpha) < 1e-10, N**2, term2)
```

```
    # I(x)
    return term1 * term2

# I(x) для квазимонохроматической волны
def intensity_quasi(x, N, b, d, L, lambda0, delta_lambda, num_wavelengths=501):
    # диапазон с центром в lambda ширины dlambda: [lambda - dlambda/2; lambda + dlambda/2] с шагом dl
    wavelengths = np.linspace(lambda0 - delta_lambda / 2, lambda0 + delta_lambda / 2, num_wavelengths
    # равномерная нормировка
    weights = np.ones_like(wavelengths)
    weights /= np.sum(weights)

    I = np.zeros_like(x)
    R = np.zeros_like(x)
    G = np.zeros_like(x)
    B = np.zeros_like(x)

    # I(x) = sum(I_mono(x, lambda_i) * weight), lambda_i in [lambda - dlambda/2; lambda + dlambda/2]
    # RGB(x) = sum(RGB_mono(x, lambda_i) * weight * I_mono(x, lambda_i)), lambda_i in [lambda - dlamb
    for wavelen, weight in zip(wavelengths, weights):
        I_mono = intensity_mono(x, N, b, d, L, wavelen)
        I += weight * I_mono

        r_c, g_c, b_c = wavelength_to_rgb(wavelen * 1e9)
        R += weight * I_mono * r_c
        G += weight * I_mono * g_c
        B += weight * I_mono * b_c

    return I, R, G, B


# Начальные параметры
N_init = 2
b_init = 1e-5
d_init = 5e-5
L_init = 1.0
lambda_init = 500e-9
lambda0_init = 500e-9
delta_lambda_init = 10e-9
mode_init = 'моно'

# Инициализация интерфейса
x = np.linspace(-0.02, 0.02, 2001)
height = 100

fig = plt.figure(figsize=(12, 10))
ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
ax2 = plt.subplot2grid((3, 1), (2, 0))
plt.subplots_adjust(left=0.1, bottom=0.4, right=0.95, top=0.95)

init_image = np.zeros((height, len(x), 3))
im = ax1.imshow(init_image, aspect='auto', extent=[x[0], x[-1], 0, height])
```

```
ax1.set_xlabel('Координата на экране (м)')
ax1.set_title('Интерференционная картина')
ax1.set_xlim(x[0], x[-1])

line, = ax2.plot(x, np.zeros_like(x), 'b-')
ax2.set_xlabel('Координата на экране (м)')
ax2.set_ylabel('Интенсивность (Вт/м^2)')
ax2.set_ylim(0, 1)
ax2.set_xlim(x[0], x[-1])
ax2.grid(True)

ax_N = plt.axes([0.25, 0.30, 0.65, 0.03])
ax_b = plt.axes([0.25, 0.25, 0.65, 0.03])
ax_d = plt.axes([0.25, 0.20, 0.65, 0.03])
ax_L = plt.axes([0.25, 0.15, 0.65, 0.03])
ax_lambda = plt.axes([0.25, 0.10, 0.65, 0.03])
ax_lambda0 = plt.axes([0.25, 0.05, 0.65, 0.03])
ax_delta_lambda = plt.axes([0.25, 0.00, 0.65, 0.03])

slider_N = Slider(ax_N, 'N', 1, 10, valinit=N_init, valstep=1)
slider_b = Slider(ax_b, 'Ширина щели (м)', 1e-6, 1e-4, valinit=b_init)
slider_d = Slider(ax_d, 'Период решетки (м)', 1e-6, 1e-4, valinit=d_init)
slider_L = Slider(ax_L, 'Расстояние до экрана (м)', 0.1, 5.0, valinit=L_init)
slider_lambda = Slider(ax_lambda, 'Длина волны (м)', 380e-9, 780e-9, valinit=lambda_init)
slider_lambda0 = Slider(ax_lambda0, 'Центр спектра (м)', 380e-9, 780e-9, valinit=lambda0_init)
slider_delta_lambda = Slider(ax_delta_lambda, 'Ширина спектра (м)', 1e-9, 100e-9, valinit=delta_lambd

rax = plt.axes([0.01, 0.05, 0.1, 0.1])
radio = RadioButtons(rax, ['моно', 'квази'], active=0 if mode_init == 'моно' else 1)

lambda_ax_pos = ax_lambda.get_position()
lambda0_ax_pos = lambda_ax_pos
dlambda_ax_pos = ax_lambda0.get_position()

# Обновление картинки при изменении параметров
def update(val):
    N = int(slider_N.val)
    b = slider_b.val
    d = slider_d.val
    L = slider_L.val

    mode = radio.value_selected

    if mode == 'моно':
        lambda_ = slider_lambda.val
        I = intensity_mono(x, N, b, d, L, lambda_)

        r_c, g_c, b_c = wavelength_to_rgb(lambda_ * 1e9)
        R = I * r_c
        G = I * g_c
        B = I * b_c
```

```
        max_val = np.max([R.max(), G.max(), B.max()])
        if max_val > 0:
            R /= max_val
            G /= max_val
            B /= max_val

        RGB = np.stack([R, G, B], axis=1)
        image = np.tile(RGB, (height, 1, 1))

        im.set_data(image)
        line.set_ydata(I)
        ax2.set_ylim(0, np.max(I) * 1.1 if np.max(I) > 0 else 1)

        ax_lambda.set_position(lambda_ax_pos)
        ax_lambda0.set_position([0, -1, 0, 0])
        ax_delta_lambda.set_position([0, -1, 0, 0])
    else:
        lambda0 = slider_lambda0.val
        delta_lambda = slider_delta_lambda.val
        I_total, R, G, B = intensity_quasi(x, N, b, d, L, lambda0, delta_lambda)

        max_val = max(np.max(R), np.max(G), np.max(B))
        if max_val > 0:
            R /= max_val
            G /= max_val
            B /= max_val

        RGB = np.stack([R, G, B], axis=1)
        image = np.tile(RGB, (height, 1, 1))

        im.set_data(image)
        line.set_ydata(I_total)
        ax2.set_ylim(0, np.max(I_total) * 1.1 if np.max(I_total) > 0 else 1)

        ax_lambda.set_position([0, -1, 0, 0])
        ax_lambda0.set_position(lambda0_ax_pos)
        ax_delta_lambda.set_position(dlambda_ax_pos)

    fig.canvas.draw_idle()

# Бинд функции обновления
slider_N.on_changed(update)
slider_b.on_changed(update)
slider_d.on_changed(update)
slider_L.on_changed(update)
slider_lambda.on_changed(update)
slider_lambda0.on_changed(update)
slider_delta_lambda.on_changed(update)
radio.on_clicked(update)

update(None)
plt.show()
```

Ссылка на репозиторий с исходным кодом: https://github.com/Sene4ka/modeling_2_sem2
Ссылка на релиз с скомпилированной программой: https://github.com/Sene4ka/modeling_2_sem2/releases/tag/v1.0.0