Фраунгоферова дифракция на отверстии произвольной формы

Исмагилов Амир, Куцубин Савва, Мельников Михаил
Группа Б02-201

5 декабря 2022 г.

Аннотация

Целью нашего проекта является численный расчёт дифракции Фраунгофера на отверстии произвольной формы. В предоставленном вашему вниманию тексте будут описанны используемые нами методы для вычисления интенсивности электромагнитного излучения в каждой точке рассматриваемой части экрана, где наблюдается диффракция, также будет показано, как осуществляется загрузка файла, содержащего необходимую информацию об отверстии, на котором наблюдается дифракция.

1 class Diffraction

1.1 Вводим параметры входной и выходной картинки, а так же необходимые физические параметры

```
# Размер исходной картинки в пикселях
self.h = h  # Высота
self.w = w  # Ширина
# Размер дифракционной картины в пикселях
self.color_grid_size = diff_grid_size
# Матрица точек отверстия исходная( картинка)
self.matrix = matrix
# Матрица дифракционной картины
self.color_matrix = np.zeros(
(self.color_grid_size, self.color_grid_size))
# Расстояние от отверстия до экрана в( микрометрах)
self.L = dist
# Физическая длина одного пикселя исходной картинки в( микрометрах)
self.pixel_len = pixel_len
# Физическая длина одного пикселя дифракционной картинки в( микрометрах)
self.pixel_len_diff = pixel_len_diff
# Длина волны в микрометрах
self.Lambda = Lambda
```

1.2 center_of_mass

Сначала предлагается рассмотреть функцию center_of_mass, которая определяет геометрический центр отверстия, считающийся точкой с нулевой фазой. Фактически центр масс вычисля-

ется по формуле

$$\vec{r_c} = \frac{\sum\limits_i m_i \vec{r_i}}{\sum\limits_i m_i}$$

если наделить каждый пиксель принадлежащий отверстию единичной массой. Ниже приведён текст соответствующей функции.

```
    def center_of_mass(self):
        '''
        Находит расположение центра масс отверстия
        Returns
        -------
        x_c : float
            Абсцисса центра отверстия.
        y_c : float
            Ордината центра отверстия.
        '''
        x_c = 0
        y_c = 0
        num_of_cells = 0
        for i in range(self.w):
            for j in range(self.h):
                if self.matrix[i][j] == 0:
                    x_c += i
                    y_c += j
                    num_of_cells += 1
        x_c /= num_of_cells
        y_c /= num_of_cells
        return (x_c, y_c)
```

Прежде чем приступить к изложению сущности двух следующих функций следует отметить, что для расчета интенсивности удобно разбить отверстие на множество квадратов и считать создаваемую ими напряженность в некоторой точке, а уже потом, посредством суммирования, получить искомую напряженность в рассматриваемой точке экрана.

1.3 summing_tension

Рассмотрим прямоугольное отверстие длиной a и b и прямоугольную систему координат с началом в центре отверстия. При Фраунгоферовой дифракции на данном отверстии напряженность поля в направлении единичного вектора $\vec{s}$ может быть вычислена по формуле:

$$E = \int\limits_{-a/2}^{a/2} \int\limits_{-b/2}^{b/2} e^{ik(s_x x + s_y y)} dxdy = ab \frac{\sin\alpha}{\alpha} \frac{\sin\beta}{\beta},\ \alpha = \frac{1}{2} k a s_x,\ \beta = \frac{1}{2} k b s_y. \tag{1}$$

s_x - Проекция единичного вектора направления дифрагированного пучка на ось OX
s_y - Проекция единичного вектора направления дифрагированного пучка на ось OY
x - координата геометрического центра отверстия (определяется как точка с нулевой начальной фазой).
y - координата геометрического центра отверстия (определяется как точка с нулевой начальной фазой).
a, b - длины сторон прямоугольника

Теперь рассмотрим произвольное препятствие, аппроксимировав его квадратной сеткой. Зная напряженность поля, создаваемого каждым квадратом сетки в рассматриваемой точке,

найдем общую напряженность поля, просуммировав поля от каждого элемента. С учетом разности фаз, получаем выражение :

$$E = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{2\pi}{\lambda} E_{0ij} \cos\left(is_x + js_y\right) \tag{2}$$

где E_{0ij} — напряженность поля от произвольного элемента сетки

Вычислением E_{0ij} по формуле (1) занимается функция calc_intensity(), вычислениями по формуле (2) — summing_tension().

```
    def summing_tension(self, s_x, s_y, E_0, x_c, y_c):
        '''
        Находит интенсивность поля в рассматриваемой точке
        Parameters
        ----------
        s_x : float
            Проекция единичного вектора направления дифрагированного пучка
            на ось 0X.
        s_y : float
            Проекция единичного вектора направления дифрагированного пучка
            на ось 0Y..
        E_0 : float
            Амплитуда напряженности поля
        x_c : float
            x - координата геометрического центра отверстия
            определяется( как точка с нулевой начальной фазой).
        y_c : float
            y - координата геометрического центра отверстия
            определяется( как точка с нулевой начальной фазой).
        Returns
        -------
        E : float
            Интенсивность в точке наблюдения P на( самом деле функция
        возвращает модуль напряженности поля в точке, но это не влияет на
        результат, а сделано в целях оптимизации и получения хорошей
        дифракционной картинки интенсивность( к краям при возведении в
        квадрат быстро убывает)).
        '''
        E = 0
        for i in range(self.w):
            for j in range(self.h):
                if self.matrix[i][j] == 0:
                    x = (i - x_c) * self.pixel_len
                    y = (j - y_c) * self.pixel_len
                    E += E_0 * \
                        np.cos((x * s_x + y * s_y)
                               * 2 * np.pi/self.Lambda)
        return abs(E)

    def calc_intensity(self):
        '''
        Подсчет интенсивности результирующего излучения в наблюдаемой точке.
        Заполняет матрицу color_matrix значениями, соответствующими
        значениям интенсивности в соответствующей точке экрана.
        Returns
```

```
        -------
        None.
        '''
        # Подсчет положения центра масс отверстия
        x_c, y_c = self.center_of_mass()
        for i in tqdm(range(self.color_grid_size)):
            for j in range(self.color_grid_size):
                # Нахождение проекций вектора направления распространения
                # дифрагированного пучка
                s_x = (i - self.color_grid_size / 2) *\
                    self.pixel_len_diff
                s_y = (j - self.color_grid_size / 2) *\
                    self.pixel_len_diff

                # Модуль вектора направления распространения
                ro = np.sqrt(s_x**2 + s_y**2 + self.L**2)

                # Конечные значения проекций единичного вектора нормировка( вектора)
                s_x /= ro
                s_y /= ro

                # alpha и beta в формуле
                alpha = np.pi * self.pixel_len * s_x / self.Lambda
                beta = np.pi * self.pixel_len * s_y / self.Lambda

                if alpha == 0:
                    a_s = 1
                else:
                    a_s = np.sin(alpha) / alpha

                if beta == 0:
                    b_s = 1
                else:
                    b_s = np.sin(beta) / beta

                E_0 = a_s * b_s  # Напряженность волны от пикселя поверхности

                # Суммарная интенсивность в каждой точке экрана
                self.color_matrix[i][j] = \
                    self.summing_tension(s_x, s_y, E_0, x_c, y_c)
```

1.4 Получение выходной картинки

Применяем определённые ранее функции для конкретного отверстия, предварительно скачав bmp-файл с изображением отверстия. В matplotlib выводим дифракционную картину двумя способами.

```
# Обработка файла
h, w, points = LoadBMP('circlehole3.bmp')

# DiffractionPicture = Diffraction(points, h, w, 150, 60, 0.555, 2 * 10**6, 10)
DiffractionPicture = Diffraction(
    points, h, w, 50, 0.555, 2 * 10**6, 100, 100)
```

```
# Подсчет интенсивности
DiffractionPicture.calc_intensity()

# График matplotlib
fig, axs = plt.subplots(1, 2, figsize=(35, 15), constrained_layout=True)
p1 = axs[0].imshow(DiffractionPicture.color_matrix,
                   cmap='hot')
fig.colorbar(p1, ax=axs[0])

p2 = axs[1].contourf(DiffractionPicture.color_matrix,
                     levels=500,
                     cmap='hot')
fig.colorbar(p2, ax=axs[1])

axs[0].set_xticks(np.linspace(0,
                              DiffractionPicture.color_grid_size, 11))
axs[0].set_xticklabels(np.linspace((-DiffractionPicture.color_grid_size/2) *
                                   DiffractionPicture.pixel_len_diff,
                                   DiffractionPicture.color_grid_size/2 *
                                   DiffractionPicture.pixel_len_diff,
                                   11), fontsize=15)
axs[0].set_yticks(np.linspace(0,
                              DiffractionPicture.color_grid_size,
                              11))

axs[0].set_yticklabels(np.linspace((-DiffractionPicture.color_grid_size/2) *
                                   DiffractionPicture.pixel_len_diff,
                                   DiffractionPicture.color_grid_size/2 *
                                   DiffractionPicture.pixel_len_diff,
                                   11), fontsize=15)
axs[1].set_xticks(np.linspace(0,
                              DiffractionPicture.color_grid_size, 11))
axs[1].set_xticklabels(np.linspace((-DiffractionPicture.color_grid_size/2) *
                                   DiffractionPicture.pixel_len_diff,
                                   DiffractionPicture.color_grid_size/2 *
                                   DiffractionPicture.pixel_len_diff,
                                   11), fontsize=15)

axs[1].set_yticks(np.linspace(0,
                              DiffractionPicture.color_grid_size,
                              11))
axs[1].set_yticklabels(np.linspace((-DiffractionPicture.color_grid_size/2) *
                                   DiffractionPicture.pixel_len_diff,
                                   DiffractionPicture.color_grid_size/2 *
                                   DiffractionPicture.pixel_len_diff,
                                   11), fontsize=15)
axs[0].set_title(
    'Дифракционная картинка, построенная с помощью функции imshow', fontsize=20)
axs[1].set_title(
    'Дифракционная картинка, построенная с помощью функции contourf', fontsize=20)

axs[0].set_xlabel('x, мкм', fontsize=17)
axs[0].set_ylabel('y, мкм', fontsize=17)
axs[1].set_xlabel('x, мкм', fontsize=17)
axs[1].set_ylabel('y, мкм', fontsize=17)
```

2 Часть программы отвечающая за загрузку файла

Импортируем библиотеки и проверяем файл на корректность.

```
import numpy as np


def error_incorrect_file():
    print('Файл BMP некорректный')
    return 0


def error_bigsize():
    print('Файл слишком большой')
    return 0
```

Функция осуществляющая загрузку файла:

```
def LoadBMP(filename):
    '''
    == ОПИСАНИЕ ФУНКЦИИ ==
    Функция получает имя файла формата BMP: filename.
    Возвращает (w, h, pixels), где w - ширина, h - высота массива,
        pixels - целочисленный массив numpy, содержащий только 0 или 1.
    Файл BMP должен быть несжатым.
    Каждый пиксель должен принимать значение (0, 0, 0) или (255, 255, 255).
    (0, 0, 0) переводится в 0 в массив numpy, (255, 255, 255) - в 1.
    В случае, если размеры файла превышают HMAX и WMAX, вызывается функция
    error_bigsize(). Если она возвращает 0, то LoadBMP() завершает работу и
    возвращает (0, 0, 0), в противном случае инчего не происходит.
    В случае, если файл не соответствует указанным выше требованиям
    или он повреждён, вызывается функция error_incorrect_file()
    и функция LoadBMP возвращает (0, 0, 0).
    '''

    HMAX = 1000
    WMAX = 1000

    header = np.zeros(27, 'int')

    file = open(filename, "rb")

    # Читаем заголовок файла из( полезной информации там w и h)
    for i in range(27):
        a = int.from_bytes(file.read(1), 'little')
        b = int.from_bytes(file.read(1), 'little')
        header[i] = b * 256 + a

    # Проверка корректности файланесжатый(, RGB, 8 бит на цвет и мб чтото- ещё)
    if (header[0] != 19778
            or header[5] != 54
            or header[7] != 40
            or header[13] != 1
            or header[14] != 24):
        file.close()
        error_incorrect_file()
```

```
        return (0, 0, 0)

    w = header[9]  # ширина
    h = header[11]  # высота
    d = w % 4  # количество пустых байт, дополняющих каждую строку в конце

    # Проверка размера файла
    if (w >= WMAX or h >= HMAX):
        if (error_bigsize() == 0):
            file.close()
            return (0, 0, 0)

    # Массив, где будет храниться картинка
    pixels = np.zeros((h, w), 'int')

    # Массив под 3 байта R, G, B
    pixel = [30] * 3

    for i in range(h):
        for j in range(w):
            for c in range(3):
                # Считываем один пиксель
                pixel[c] = int.from_bytes(file.read(1), 'little')

    # Записысаем подходящее значение в файл, проверяя соответствие формату
            if (pixel[0] == pixel[1] and pixel[1] == pixel[2]):
                if (pixel[0] == 0):
                    pixels[h - i - 1][j] = 0
                elif (pixel[0] == 255):
                    pixels[h - i - 1][j] = 1
                else:
                    file.close()
                    error_incorrect_file()
                    return (0, 0, 0)
            else:
                file.close()
                error_incorrect_file()
                return (0, 0, 0)

        # Пропускаем пустые байты
        if d != 0:
            for i in range(d):
                file.read(1)

    file.close()
    return(w, h, pixels)
    # Конец функции LoadBMP()
```

3 Samples

Здесь приведены bmp-картинки переведённые в массив numpy с помощью функции LoadBMP(). Эти примеры понадобятся далее в качестве демонстрации работы программы. Ввиду немалого размера массивов приведем лишь одну из четырёх функций, содержащую информацию от

круговом отверстии.

```
def Example1():
    w = 40
    h = 40
    image = [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
        0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
```

```
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
     0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

return w, h, image
```

4 Интерфейс программы

Для запуска этого блока программы следует предварительно запустиь все блоки приведенные выше. Именно эта часть кода отвечает за интерфейс программы. Окно на котором будет представлен результат работы программы содержит следующие элементы:

- Изображение отверстия на котором планируется налюдать дифракци.
- Дифракционную картину.
- Так же предоставлена возможность выбора физических параметров системы и масштаба картинки (значения могут быть введены непосредственно в части окна находящейся под изображением отверстия).
- Постредством нажатия кнопки "Вычислить дифракционную картину"можно запустить расчет дифракционной картины, процент выполненной работы в любой момент времени можно оценить по индикатору загрузки.
- В левом верхнем углу окна присутствует кнопка "файл". При наведении курсора на эту кнопку пользователю будет предложено загрузить в программу подготовленную ранее

картинку или же рассмотреть дифракцию на отверстиях встроенных в программе (Примеры 1-4)

```
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox as mb
import tkinter.ttk as ttk

from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
                                               NavigationToolbar2Tk)

from BMP import *
from Diffraction import *
from Samples import *

def NewProject():
    AskSampleWindow = tk.Toplevel(window)
    AskSampleWindow.geometry('250x250')
    AskSampleWindow.title('Выберите пример')

    tk.Label(AskSampleWindow,
             text = 'Выберите пример для вычисления\n дифракционной картины').pack()

    var = tk.IntVar()
    var.set(0)
    ex1 = tk.Radiobutton(AskSampleWindow, text = "Круговое отверстие", variable = var,
    value = 0)
    ex2 = tk.Radiobutton(AskSampleWindow, text = "Шестиугольное отверстие", variable =
    var, value = 1)
    ex3 = tk.Radiobutton(AskSampleWindow, text = "Отверстие в виде звёздочки", variable =
    var, value = 2)
    ex4 = tk.Radiobutton(AskSampleWindow, text = "Дифракционная решётка", variable = var,
    value = 3)
    ex1.pack()
    ex2.pack()
    ex3.pack()
    ex4.pack()
    button = tk.Button(AskSampleWindow, text = "Открыть", command = lambda: Sample(var,
    AskSampleWindow))
    button.pack()

    '''w, h, image = Example3()
    ObstacleLoaded(w, h, image, 555, 200, 2, 60)'''

def Sample( number, askwindow ):
    askwindow.destroy()
    if (number.get() == 0):
        w, h, image = Example1()
    elif (number.get() == 1):
        w, h, image = Example2()
    elif (number.get() == 2):
        w, h, image = Example3()
    else:
        w, h, image = Example4()
```

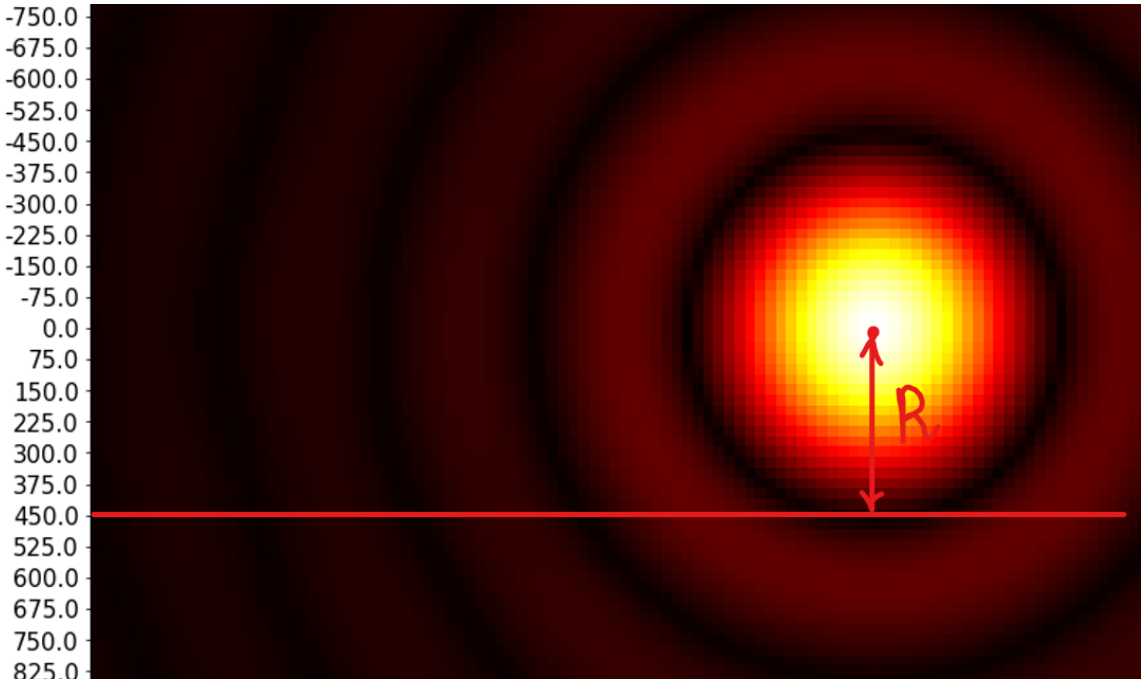
```
    ObstacleLoaded(w, h, image, 555, 200, 2, 60, 15)

def OpenProject():
    tk.messagebox.showinfo('Ой', 'А вот это я не сделал...')

def LoadBMPFile():
    # Запрос на открытие картинки
    filename = filedialog.askopenfilename()
    # Если юзер выбрал файл, то обрабатываем его
    if (filename != ''):
        w, h, image = LoadBMP(filename)

        if w > 0 and h > 0: # Файл открылся успешно
            ObstacleLoaded(w, h, image, 555, 200, 2, 60, 10)

def ObstacleLoaded(w, h, image, WV, Size, L, Scale, PixelL):
    РИСУЕМ#== ПРЕПЯТСТВИЕ==
    fig = Figure(figsize = (4, 3), dpi = 100)
    plot = fig.add_subplot(111)

    # Рисуем картинку
    plot.imshow(image, cmap = 'gray')
    plot.grid(False)
    plot.set_xticks([])
    plot.set_yticks([])
    fig.suptitle('Препятствие')

    # Это будет нарисовано в окне
    canvas = FigureCanvasTkAgg(fig, master = window)
    canvas.draw()

    # Рисуем
    canvas.get_tk_widget().grid(row = 0, column = 0, columnspan = 2)

    # Длина волны
    WV_enter = tk.Entry(width = 6)
    WV_enter.grid(row = 1, column = 1, sticky='W')
    WV_enter.insert(0, str(WV))

    WV_label = tk.Label(text = 'Длина волны, нм')
    WV_label.grid(row = 1, column = 0, sticky='W')

    # Разрешение диф картинки
    Size_enter = tk.Entry(width = 6)
    Size_enter.grid(row = 2, column = 1, sticky='W')
    Size_enter.insert(0, str(Size))

    Size_label = tk.Label(text = 'Разрешение дифракционной картинки, px')
    Size_label.grid(row = 2, column = 0, sticky='W')

    # Расстояние от препятствия до экрана
    L_enter = tk.Entry(width = 6)
    L_enter.grid(row = 3, column = 1, sticky='W')
    L_enter.insert(0, str(L))
```

```
    L_label = tk.Label(text = 'Расстояние от препятствия до экрана, м')
    L_label.grid(row = 3, column = 0, sticky='W')

    # Масштаб диф картинки
    Scale_enter = tk.Entry(width = 6)
    Scale_enter.grid(row = 4, column = 1, sticky='W')
    Scale_enter.insert(0, str(Scale))

    Scale_label = tk.Label(text = 'Масштаб картинки')
    Scale_label.grid(row = 4, column = 0, sticky='W')

    # Размер пикселя
    PixelL_enter = tk.Entry(width = 6)
    PixelL_enter.grid(row = 5, column = 1, sticky='W')
    PixelL_enter.insert(0, str(PixelL))

    PixelL_label = tk.Label(text = 'Размер пикселя, мкм(?)')
    PixelL_label.grid(row = 5, column = 0, sticky='W')

    tk.Button(text = 'Вычислить дифракционную картину',
              command = lambda: ComputeDifraction(image, h, w, WV_enter, Size_enter,
   L_enter, Scale_enter, PixelL_enter)).grid(row = 6, column = 0)



def ComputeDifraction(points, h, w, WV_enter, Size_enter, L_enter, Scale_enter,
   PixelL_enter):
    Wavelength = float(WV_enter.get()) / 1000 # Перевели в мкм
    Size = int(Size_enter.get())
    L = float(L_enter.get()) * 10**6 # Перевели в мкм
    Scale = int(Scale_enter.get())
    PixelLen = int(PixelL_enter.get())

    progress_label = tk.Label(text = 'Прогресс выполнения:')
    progress_label.grid(row = 1, column = 2)
    progress_bar = ttk.Progressbar(window, orient = "horizontal", mode = "determinate",
                                   maximum = 100, value = 0, length = 200)
    progress_bar.grid(row = 2, column = 2)


    DiffractionPicture = Diffraction(points, h, w, Size, Scale, Wavelength, L, PixelLen,
   progress_bar, window)
    DiffractionPicture.calc_intensity()

    # Рисуем картинку
    fig = Figure(figsize = (4, 3), dpi = 100)
    plot = fig.add_subplot(111)

    plot.imshow(DiffractionPicture.color_matrix, cmap = 'hot')
    plot.grid(False)
    plot.set_xticks([])
    plot.set_yticks([])
    fig.suptitle('Дифракционная картина')

    # Это будет нарисовано в окне
```

```
    canvas = FigureCanvasTkAgg(fig, master = window)
    canvas.draw()

    # Рисуем
    canvas.get_tk_widget().grid(row = 0, column = 2)

    progress_bar.grid_forget()
    tk.Label(text='Завершено').grid(row = 2, column = 2)Создаём


# окно
window = tk.Tk()
window.title("Моделирование дифракции v0.1")
window.geometry('800x600')

#== МЕНЮ ==
menu = tk.Menu(window)Файл

#
Menu_File = tk.Menu(menu, tearoff = 0)

Menu_File.add_command(label = 'Новый проект', command = NewProject)
Menu_File.add_command(label = 'Создать проект из файла BMP', command = LoadBMPFile)
Menu_File.add_separator()
Menu_File.add_command(label = 'Загрузить проект', command = OpenProject)


menu.add_cascade(label = 'Файл', menu = Menu_File)
window.config(menu = menu)

w, h, img = Example1()
ObstacleLoaded(w, h, img, 555, 200, 2, 60, 15)

#== МЕНЮ закончилось ==

window.mainloop()
```

5 Сравнение параметров полученных дифракционных картин с теоретическими значениями

Рис. 1: Дифракционная картина полученная на круговом отверстии



Как можно видеть по Рис. 1 первый дифракционный минимум наблюдается на расстоянии 450 мкм от центра картины.
Приближенно угловые радиусы тёмных колец могут быть вычислены по следующей формуле:

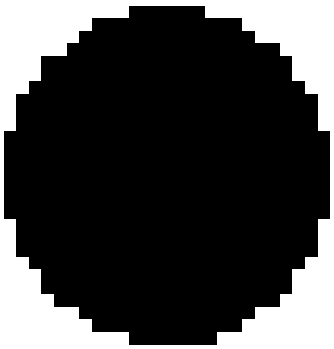
$$\vartheta_m = \left[0.61 + \frac{m-1}{2}\right] \frac{\lambda}{R}$$

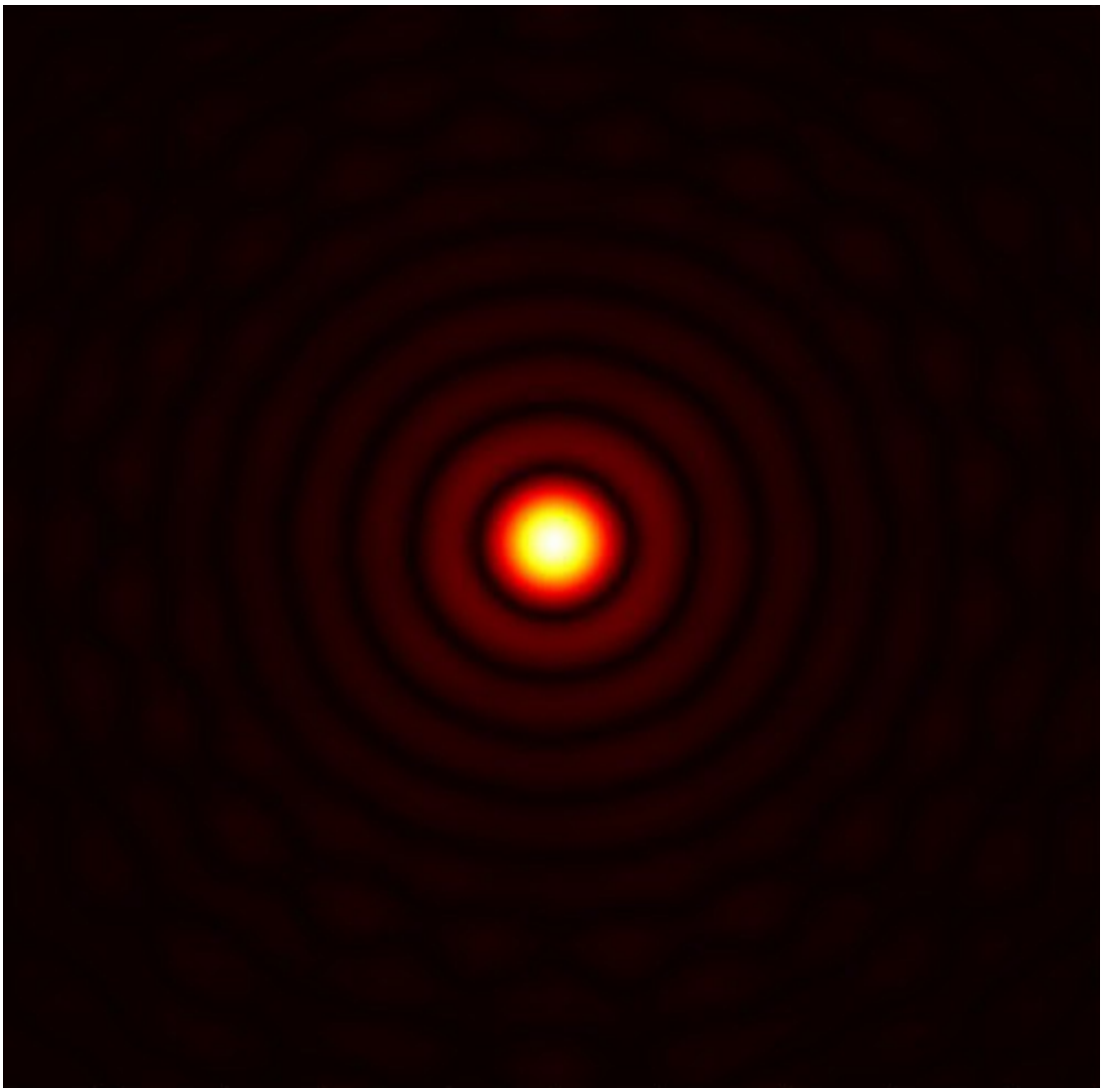
- m - номер минимума
- $\lambda = 555$ нм - длина волны
- $R = 1500$ мкм - Радиус отверстия

Для вычисления теоретического значения радиуса темных колец ещё необходимо знать расстояние от отверстия до экрана, в нашем случае $L = 2$ м. Тогда радиус первого тёмного кольца равен 451.4 мкм, с учётом приближенности значения можно считать, что значения совпадают, и программу можно считать точной.

6 Примеры

6.1





6.2