Отчет по лабораторной работе №3 по оптической информатике

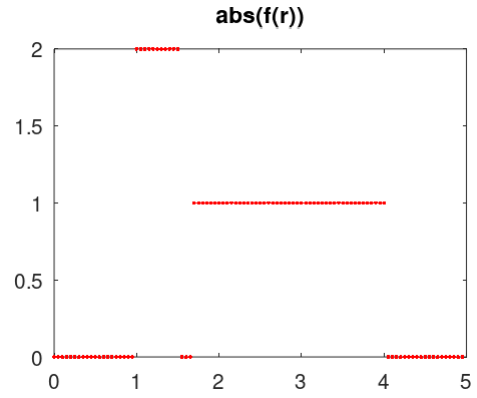
Выполнил: Чичикин Артем

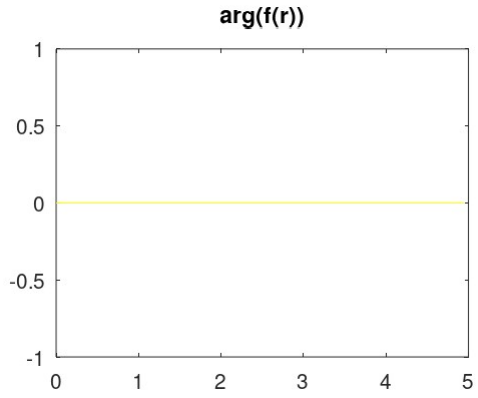
Проверил: Кириленко Михаил Сергеевич

Исходный код: https://github.com/ArtyomStebenev/OI

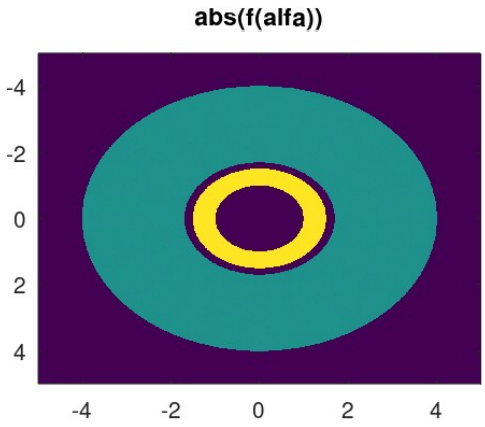
Первоначально создаю файл **get_function_indicator.m**, где лежит одноименная функция. В зависимости от значений входного вектора: единица, если оно принадлежит ли интервалу, задаваемому вторым и третьим аргументами функции (числами), и ноль в противном случае.

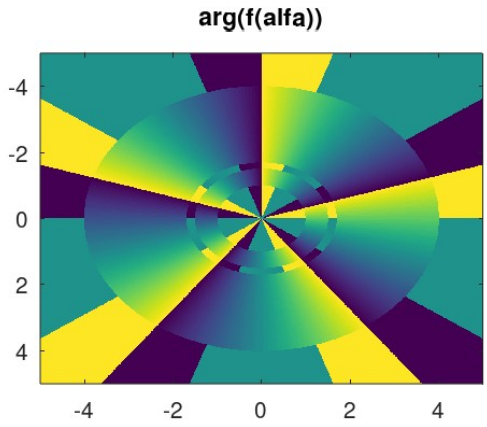
Далее создаю **main.m**, где будут лежать константы и вызываться функции. Инициализирую переменные `r` и `f`.



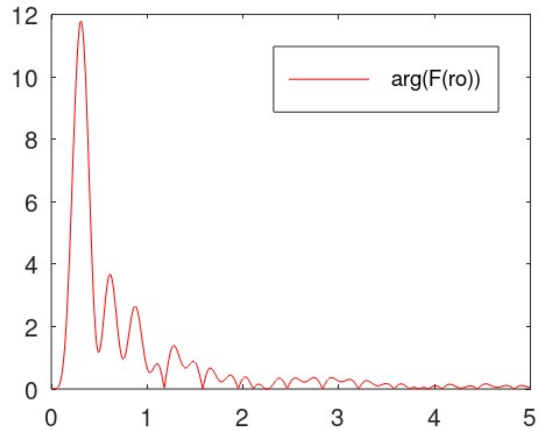


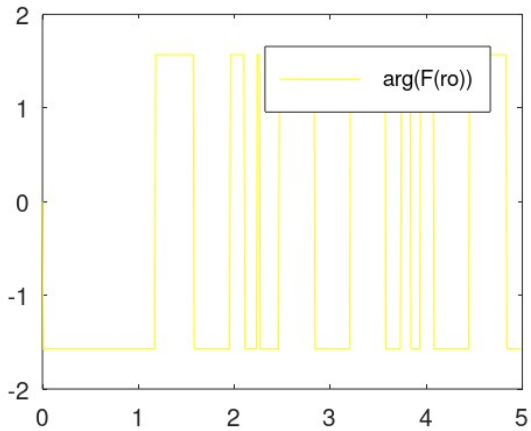
Во избежании бойлерплейт кода, создаю функцию `get_alfa_matrix`. Исходя из алгоритма, нахожу матрицу `alfa` и, если индекс получился больше, чем n, то зануляю элементы. Сама функция возвращает `r_alfa`. Нахожу радиально-вихревую функцию `f_alfa` и вывожу ее на экран. Нахождением угла занимается функция `get_phi`.



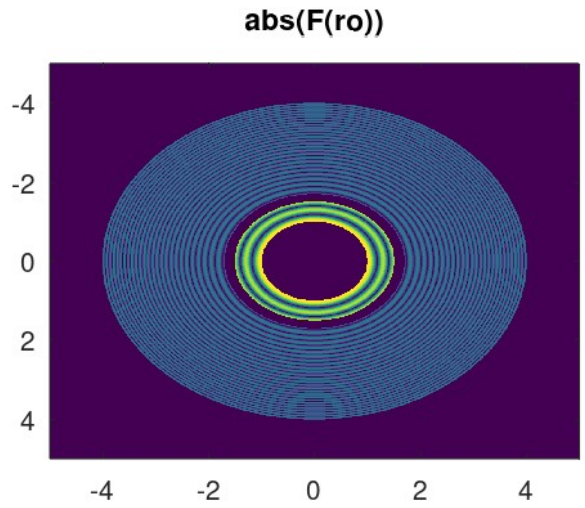


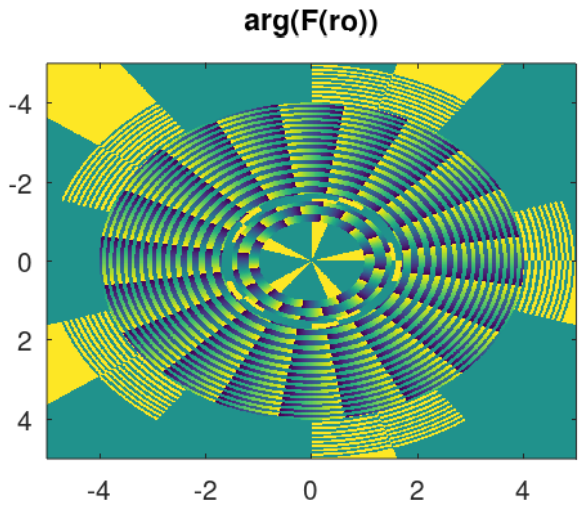
Далее создаю отдельный файл **hankel_transform_sript.m**, для дальнейшей работы с преобразованием Ханкеля. Создаю переменную `ro` — дискретный аналог частоты. Для вычисления интеграла использую те же приемы, что и во второй лабораторной работе. Вывожу преобразование Ханкеля от исходной функции.





Нахожу радиально-вихревую функцию `F_alfa` и вывожу ее на экран. Можно заметить, что количество вершин у звезды аргумента соответствует значению m.





Листинг 1.

```
# Consts
n = 500;
R_CONST = 5;
step_r = R_CONST/n;

r1 = 1;
r2 = 1.5;
r3 = 1.7;
r4 = 4;
h1 = 2;
h2 = 1;
m = 5;



# 5
r = step_r .* ([1:n] - 1);
f = h1 * get_function_indicator(r, r1, r2) + h2 *
get_function_indicator(r, r3, r4);

plot(r(1:5:end), abs(f(1:5:end)), ".r");
title("abs(f(r))");

figure;
plot(r(1:5:end), arg(f(1:5:end)), "-y");
title("arg(f(r))");


# 8
r_alfa = get_alfa_matrix(n, R_CONST);


f_alfa = (h1 * get_function_indicator(r_alfa, r1, r2)
       +  h2 * get_function_indicator(r_alfa, r3, r4));
f_alfa .*= exp(i*m * get_phi(n));
```

```
figure;
imagesc(-R_CONST:step_r:R_CONST, -R_CONST:step_r:R_CONST, abs(f_alfa));
title("abs(f(alfa))");

figure;
imagesc(-R_CONST:step_r:R_CONST, -R_CONST:step_r:R_CONST, arg(f_alfa));
title("arg(f(alfa))");
```

Листинг 2.

```
P_CONST = 5;
step_ro = P_CONST/n;

# 9
ro = 0:step_ro:P_CONST-step_ro/2;

Kernel = besselj(m, 2*pi* r .* ro') .* r;
F_ro = (2*pi/i^m) * Kernel * f.' * step_r;
F_ro = F_ro.';

plot(ro, abs(F_ro), "-r; arg(F(ro));");
figure;
plot(ro, arg(F_ro), "-y; arg(F(ro));");


# 12
ro_alfa = get_alfa_matrix(n, P_CONST);

Kernel_matrix = besselj(m, 2*pi * r_alfa .* ro_alfa) .* r_alfa;

F_alfa = (2*pi/i^m) * Kernel_matrix .* f_alfa.' * step_ro;
F_alfa = F_alfa.';
F_alfa .*= exp(i*m * get_phi(n));



figure;
imagesc(-P_CONST:step_ro:P_CONST, -P_CONST:step_ro:P_CONST, abs(F_alfa));
title("abs(F(ro))");

figure;
imagesc(-P_CONST:step_ro:P_CONST, -P_CONST:step_ro:P_CONST, arg(F_alfa));
title("arg(F(ro))");
```

Листинг 3.

```
function result_vector = get_function_indicator(input_vector, left_border, right_border)
    result_vector = (input_vector >= left_border & input_vector <= right_border) * 1;
endfunction

# "*1" is needed for transform from bool to double
```

Листинг 4.

```
function phi = get_phi(n)
  k = [1:2*n-1];
  j = k';
  phi = atan2(k-n, j-n);
endfunction
```

Листинг 5.

```
function result_matrix = get_alfa_matrix(n, BORDER)
  step = BORDER/n;

  k = [1:2*n-1];
  j = k';


  alfa = zeros(2*n-1, 2*n-1);
  alfa = round(sqrt((j-n).^2 + (k-n).^2)) + 1;
  alfa = (alfa <= n) .* alfa; % cutting the unnessessery

  alfa_matrix = (alfa-1) * step;

  result_matrix = alfa_matrix;
endfunction
```