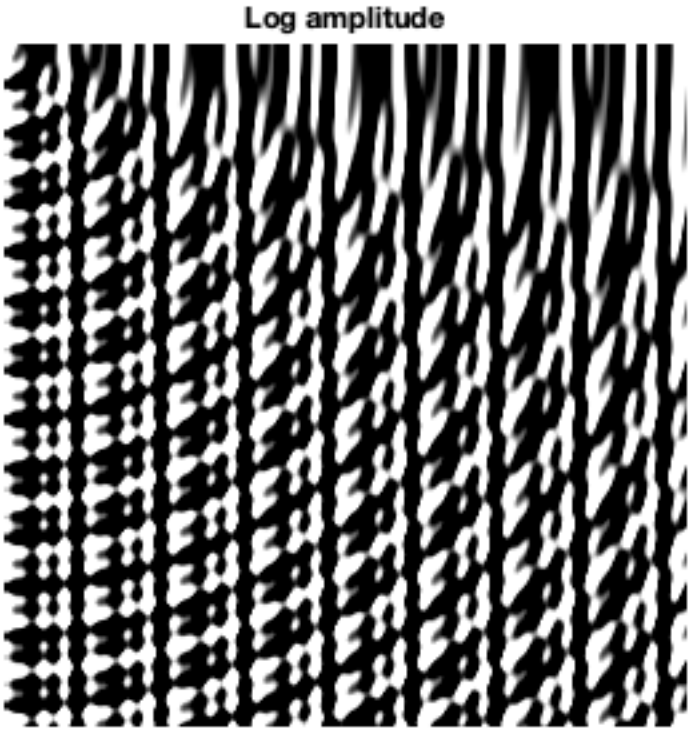
Part 1: Discrete Fourier Transform

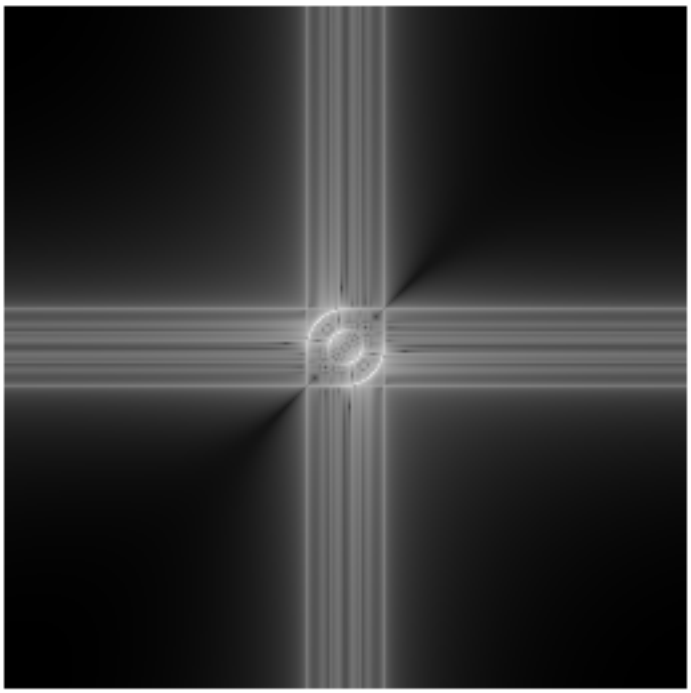
```
% a
% a step 1
rows = 512;
[x,y] = meshgrid(1:rows, 1:rows);

% a step 2
img = sin(0.2 * x) + sin(0.3 * x) + cos(0.4 * x) + sin(sqrt(x .* x +
 y .* y) * 0.15) + sin(sqrt(x .* x + y .* y) * 0.35);
figure;
imshow(img);
title("Original image");

% a step 3
fftImg = fftshift(fft2(img));
% Show the magnitude and phase of DFT for this image
phase = angle(fftImg);
amplitude = abs(fftImg);

figure;
imshow(log(amplitude),[]);
title("Log amplitude");

figure;
imshow(unwrap(phase), []);
title("Phase");

% a step 4
% Multiply the magnitude of DFT with 2
mul = ifftshift(amplitude) * 2;
% Calculate the inverse Discrete Fourier Transform
ifftImg = ifft2(mul);

figure;
imshow(ifftImg);
title("Magnitude multiply 2");

% Explain:
% After multiplying the magnitude of DFT with 2 and applying inverse
 IDF,
% the high frequency pattern increased compared with the original
 image.

% b
img = im2double(imread('./images/Cross.jpg'));

dft = fft2(img);
fftImg = fftshift(dft);

figure;
imshow(fftImg);
```

```
% Explain:
% The diagonal line pattern in orginal image also exists in the DFT
 image.
```

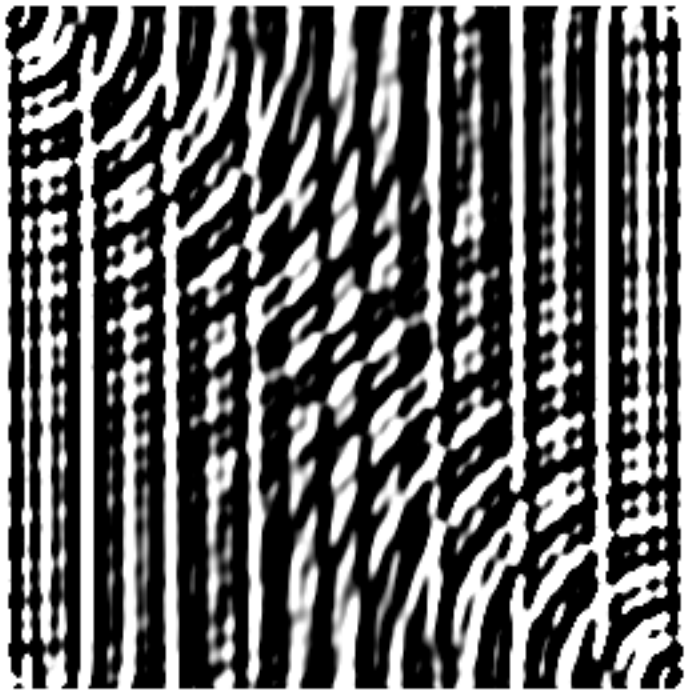
Warning: Displaying real part of complex input.

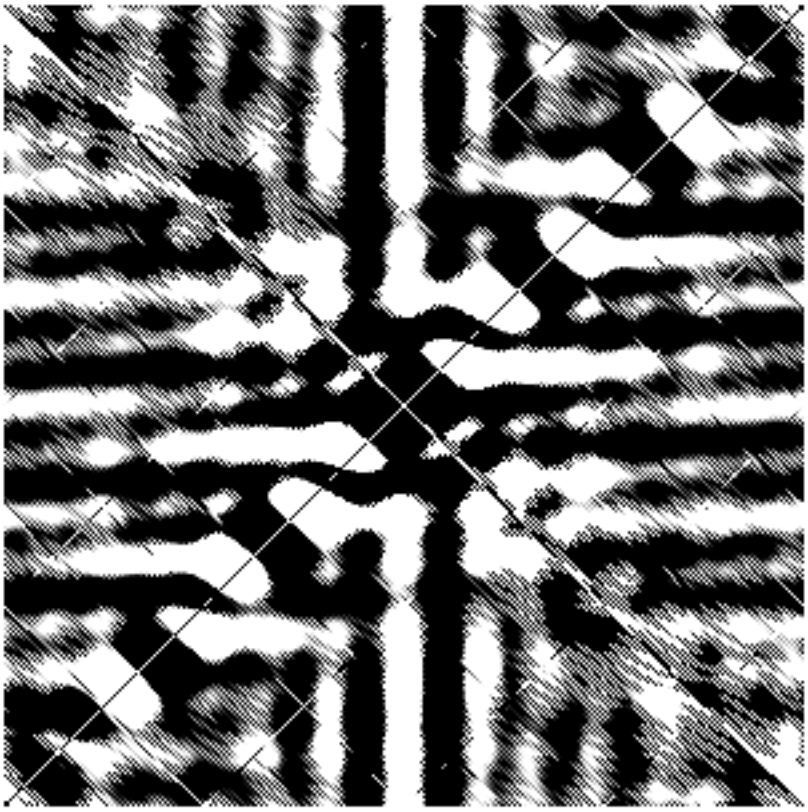


Phase



Magnitude multiply 2





Published with MATLAB® R2019b

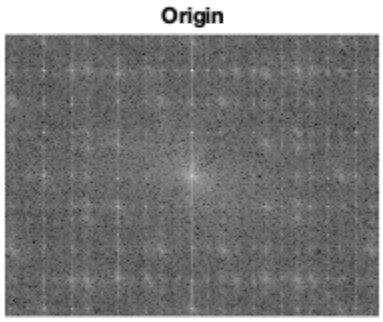
Part 2: Notch Filter

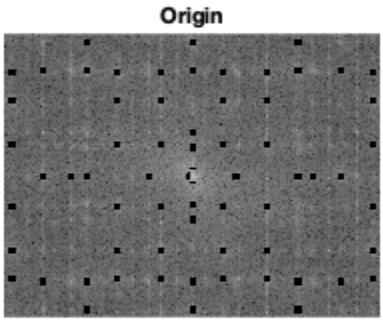
Image: moonlanding

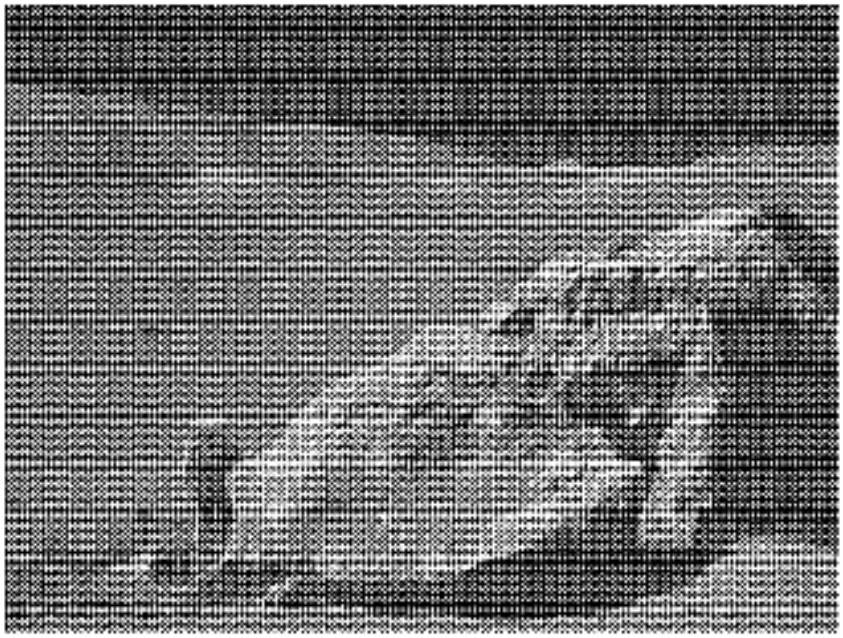
```
image = (imread("./images/moonlanding.png"));
image = im2double(image);

% Apply notch filter
filteredImg = notchFilter(image, [0.5, 1]);

figure;
imshow(image);
title('Origin');

figure;
imshow(filteredImg, []);
title('Noise removed');

% Image: psnr2

image = rgb2gray(imread("./images/psnr2.png"));
image = im2double(image);

% Apply notch filter
filteredImg = notchFilter(image,[0.75, 1]);

figure;
imshow(image);
title('Origin');

figure;
imshow(filteredImg, []);
title('Noise removed');

function res = notchFilter(image, range)
    % Calcualte DFT
    fftImg = fftshift(fft2(image));
    amp = abs(fftImg);

    minAmp = min(amp(:));
    s = sort(amp(:));
    maxAmp = s(end - 20);

    % Calculate lower bound and upper bound
    lowerbound = (maxAmp - minAmp) * range(1) + minAmp;
    upperbound = (max(amp(:)) - minAmp) * range(2) + minAmp;
    band = (amp >= lowerbound) & (amp < upperbound);

    peak = (amp == ordfilt2(amp, 9 , ones(3, 3))) & band;
    [w, h] = size(image);
    [r, c] = find(peak);
```

```
    removeLength = 5;

    noise = zeros(size(fftImg));
    for i = 1:length(r)
        % Edge case
        if (w-r(i))^2+(h-c(i))^2 <= removeLength^2
            continue;
        end
        if r(i) <= removeLength || c(i) <= removeLength
            continue;
        end
        % Calculate the end of rows and cols
        if w < r(i) + removeLength
            rowEnd = w;
        else
            rowEnd = r(i) + removeLength;
        end
        if h < c(i) + removeLength
            colEnd = h;
        else
            colEnd = c(i) + removeLength;
        end

        dr = r(i) - removeLength:rowEnd;
        dc = c(i) - removeLength:colEnd;

        noise(dr, dc) = 1;
    end

    % Get the central of the image
    cx = round((size(noise, 2) + 1) / 2);
    cy = round((size(noise, 1) + 1) / 2);

    dcr = cy - 2*removeLength:cy + 2*removeLength;
    drr = cx - removeLength:cx + 2*removeLength;
    noise(dcr, drr) = 0;

    fftImg(noise > 0) = 0;

    showResult(amp, fftImg);

    res = real(ifft2(fftshift(fftImg)));
end

function result = showResult(amp, fftImg)
    figure;

    subplot(121)
    imshow(log(amp),[]);
    title("Origin");

    subplot(122)
    imshow(real(log(abs(fftImg))), []);
    title('Noise removed');
```

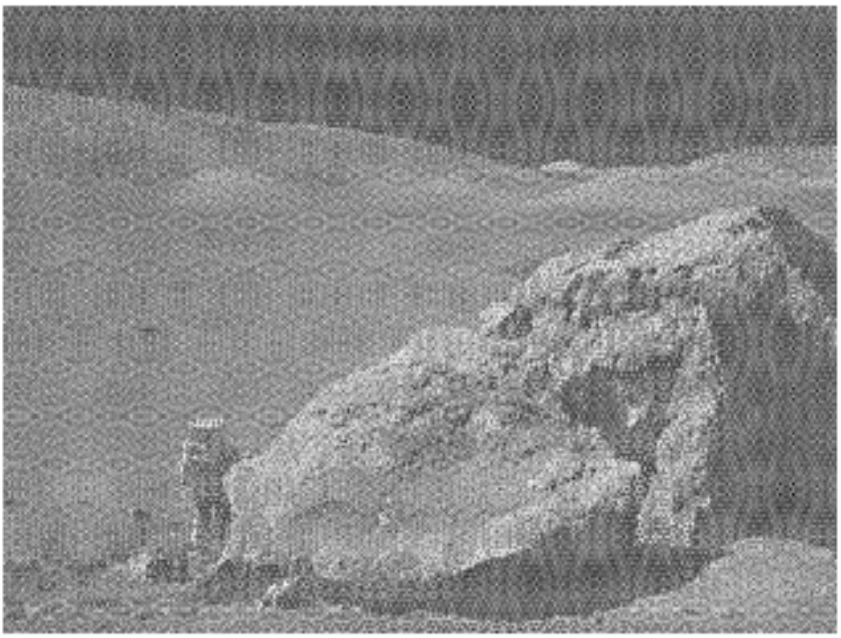
```
end
```



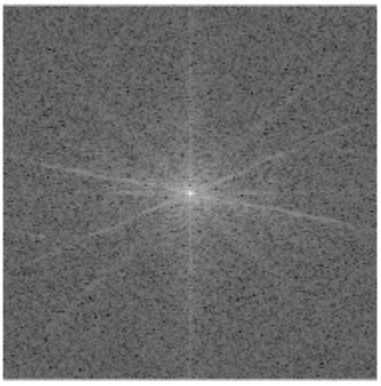




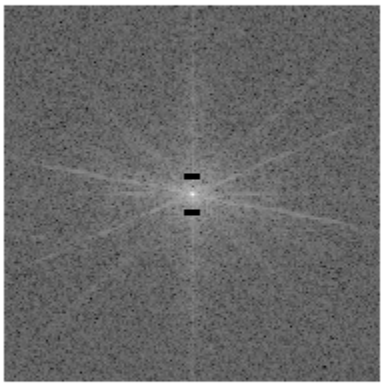
Noise removed



Origin



Noise removed

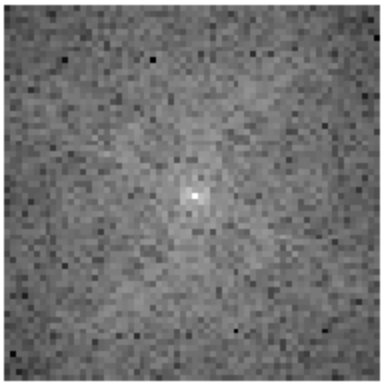


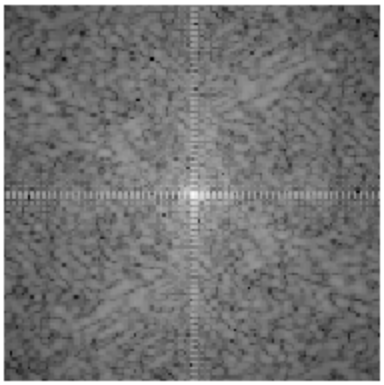
Published with MATLAB® R2019b

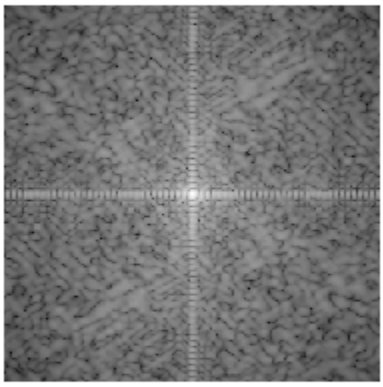
Part 3: Analyzing DFT

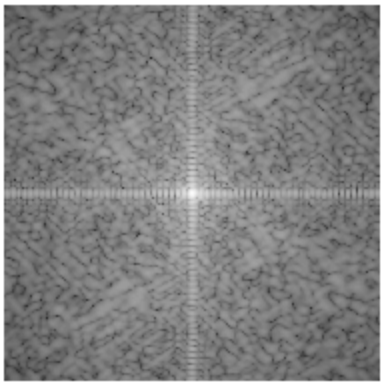
```
% Choose a 64x64 image and find the Discrete Fourier Transform for the
 image
image64 = imresize(imread("../images/CARTOON.jpg"), [64,64]);

figure;

subplot(121)
imshow(image64)

subplot(122)
imshow(log(abs(fftshift(fft2(image64)))), []);


% Add 64 columns and rows of zeros to the right and bottom side of the
 original image
[rows, cols] = size(image64);

image128 = zeros(2 * rows, 2 * cols);
image128(1:rows,1:cols) = image64;

figure;

subplot(121)
imshow(image128);

subplot(122)
imshow(log(abs(fftshift(fft2(image128)))), []);

%  Repeat this process 2
[rows, cols] = size(image128);

image256 = zeros(2 * rows, 2 * cols);
image256(1:rows,1:cols) = image128;

figure;

subplot(121)
imshow(image256);

subplot(122)
imshow(log(abs(fftshift(fft2(image256)))), []);

% Repeat this process 2
[rows, cols] = size(image256);

image512 = zeros(2 * rows, 2 * cols);
image512(1:rows,1:cols) = image256;

figure;
```

```
subplot(121)
imshow(image512);

subplot(122)
imshow(log(abs(fftshift(fft2(image512)))), []);
```









Published with MATLAB® R2019b

Part 2: Notch Filter

Image: moonlanding

```
image = (imread("./images/moonlanding.png"));
image = im2double(image);

% Apply notch filter
filteredImg = notchFilter(image, [0.5, 1]);

figure;
imshow(image);
title('Origin');

figure;
imshow(filteredImg, []);
title('Noise removed');

% Image: psnr2

image = rgb2gray(imread("./images/psnr2.png"));
image = im2double(image);

% Apply notch filter
filteredImg = notchFilter(image,[0.75, 1]);

figure;
imshow(image);
title('Origin');

figure;
imshow(filteredImg, []);
title('Noise removed');

function res = notchFilter(image, range)
    % Calcualte DFT
    fftImg = fftshift(fft2(image));
    amp = abs(fftImg);

    minAmp = min(amp(:));
    s = sort(amp(:));
    maxAmp = s(end - 20);

    % Calculate lower bound and upper bound
    lowerbound = (maxAmp - minAmp) * range(1) + minAmp;
    upperbound = (max(amp(:)) - minAmp) * range(2) + minAmp;
    band = (amp >= lowerbound) & (amp < upperbound);

    peak = (amp == ordfilt2(amp, 9 , ones(3, 3))) & band;
    [w, h] = size(image);
    [r, c] = find(peak);
```

```
    removeLength = 5;

    noise = zeros(size(fftImg));
    for i = 1:length(r)
        % Edge case
        if (w-r(i))^2+(h-c(i))^2 <= removeLength^2
            continue;
        end
        if r(i) <= removeLength || c(i) <= removeLength
            continue;
        end
        % Calculate the end of rows and cols
        if w < r(i) + removeLength
            rowEnd = w;
        else
            rowEnd = r(i) + removeLength;
        end
        if h < c(i) + removeLength
            colEnd = h;
        else
            colEnd = c(i) + removeLength;
        end

        dr = r(i) - removeLength:rowEnd;
        dc = c(i) - removeLength:colEnd;

        noise(dr, dc) = 1;
    end

    % Get the central of the image
    cx = round((size(noise, 2) + 1) / 2);
    cy = round((size(noise, 1) + 1) / 2);

    dcr = cy - 2*removeLength:cy + 2*removeLength;
    drr = cx - removeLength:cx + 2*removeLength;
    noise(dcr, drr) = 0;

    fftImg(noise > 0) = 0;

    showResult(amp, fftImg);

    res = real(ifft2(fftshift(fftImg)));
end

function result = showResult(amp, fftImg)
    figure;

    subplot(121)
    imshow(log(amp),[]);
    title("Origin");

    subplot(122)
    imshow(real(log(abs(fftImg))), []);
    title('Noise removed');
```

```
end
```

Published with MATLAB® R2019b