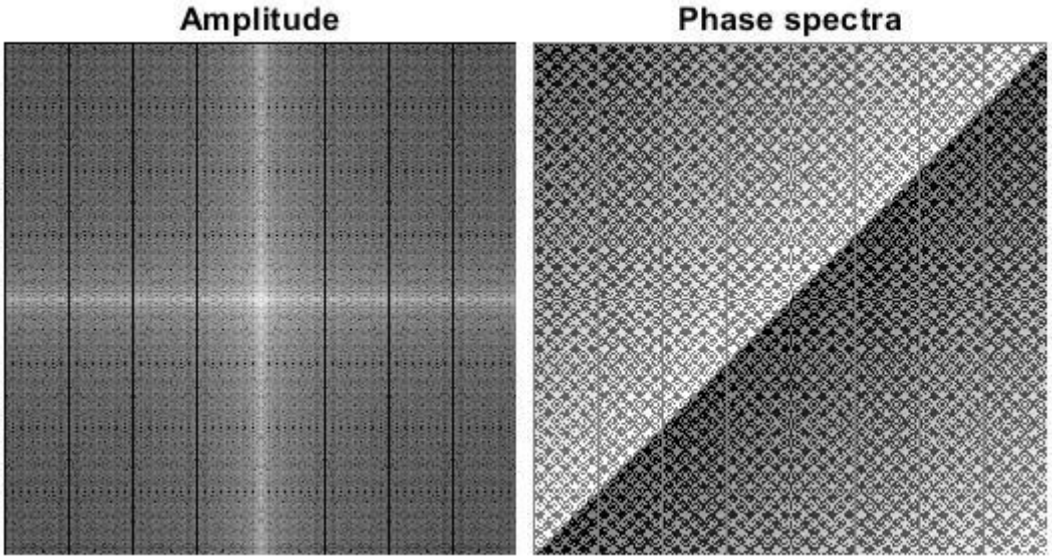
ธนาคม หัสแดง 590610624

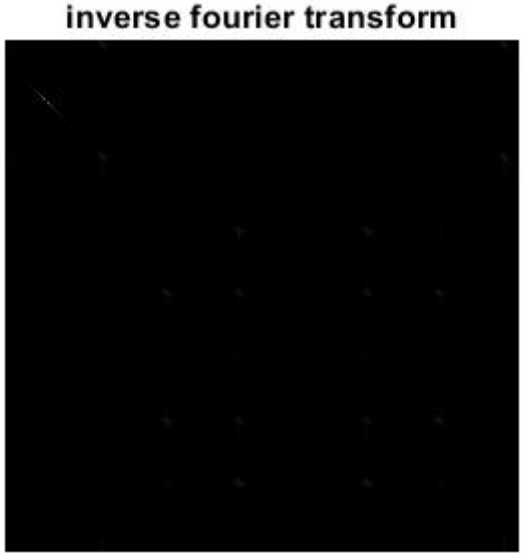
Computer Assignment 2

1. Properties of the Fourier Transform

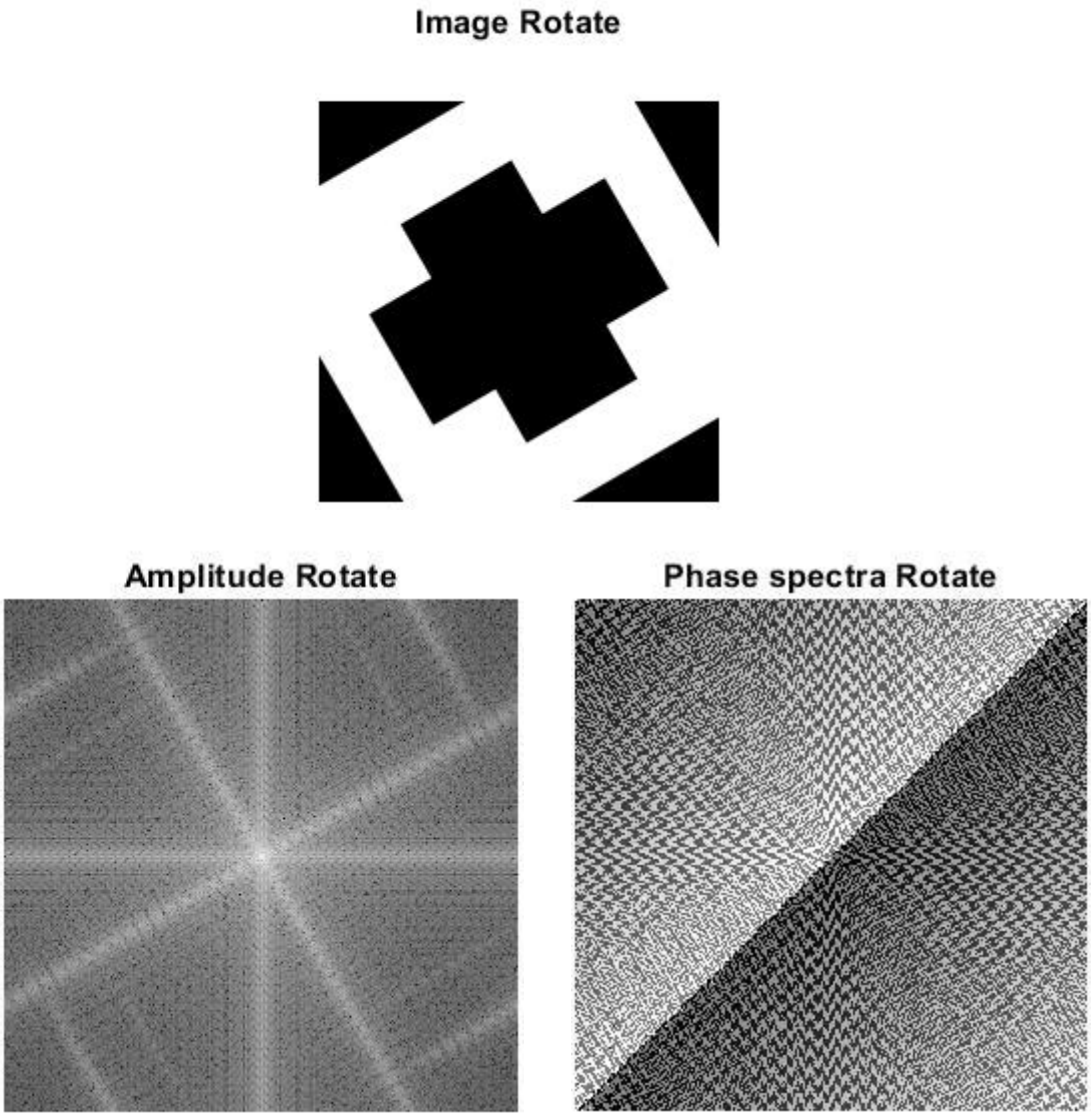
1.1 ทำการ Pad รูปจากเดิมที่ขนาด 200x200 ให้เป็น 256x256 หลังจากนั้นทำ FFT และ Shift ในการหา Amplitude ให้นำค่า shift ของ FFT ที่ได้ไป abs และ log ตามลำดับ ในกรณีของ Phase ให้นำ Shift ของ FFT ไปเข้าฟังก์ชั่น angle ก็จะได้รูปของ Phase ตามต้องการ



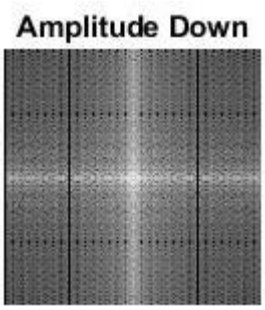
1.2 นำ Phase ของข้อ 1.1 มาคูณกับ complex ที่มีค่า exp(-2j*pi*(((20*X)/256)+((30*Y)/256))) แล้วไปนำไปเข้ากระบวนการ IFFT คือ IFFTShift และ IFFT ตามลำดับ แล้วจึงนำไปหา abs ก็จะได้รูปมุมของรูป โดยรูปจะมีการเลื่อนแกนไป x = 20 , y = 30 ดังรูป

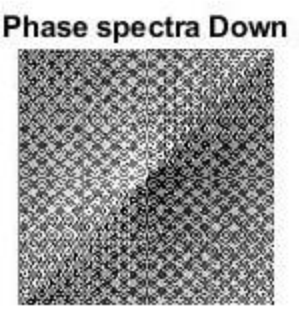


1.3 นำรูปที่ต้องการมาหมุนด้วยมุม 30 องศา หลังจาดนั้นก็เข้ากระบวนการ FFT เพื่อหา Amplitude และ Phase เหมือนข้อ 1.1 โดยผลลัพธ์ จะเห็นว่า รูปของ Amplitude นั้น เมื่อเทียบกับกรณีที่ยังไม่ได้หมุน รูปผลลัพธ์ที่ได้จะ หมุนไปในทิศทางเดียวกันกับที่เราหมุนรูป โดยได้ผลลัพธ์ดังรูป



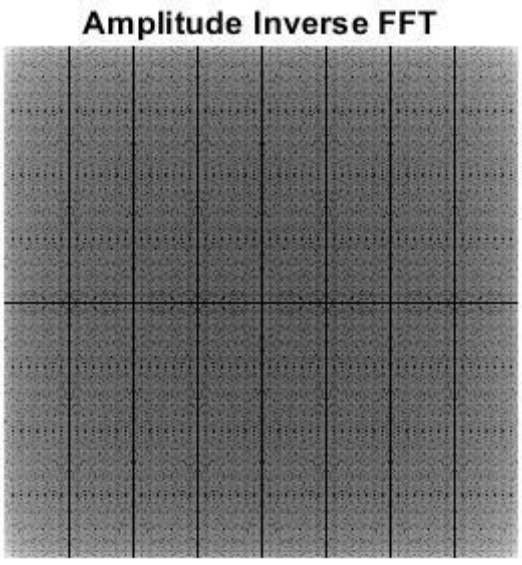
1.4 นำรูปที่ต้องการมาลดขนาดให้เท่าที่เราต้องการ ในกรณีของข้อนี้ ให้ลด ½ แล้วนำรูปที่ได้ไป Pad และทำกระบวนการเพื่อหา Amplitude และ Phase ตามข้อ 1.1 โดยผลลัพธ์ จะเห็นว่า รูป Amplitude และ Phase นั้น ความละเอียดจะลดลง ขนาดของรูปก็เช่นกัน แต่ในส่วนของรายละเอียดอื่น ๆ แทบไม่ได้แตกต่างกันเลย ดังรูป

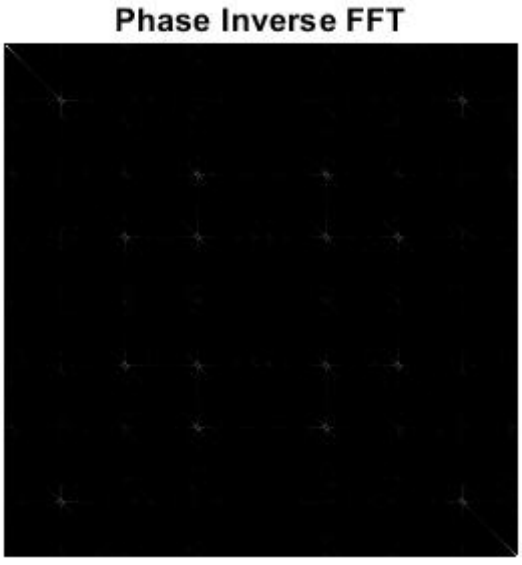




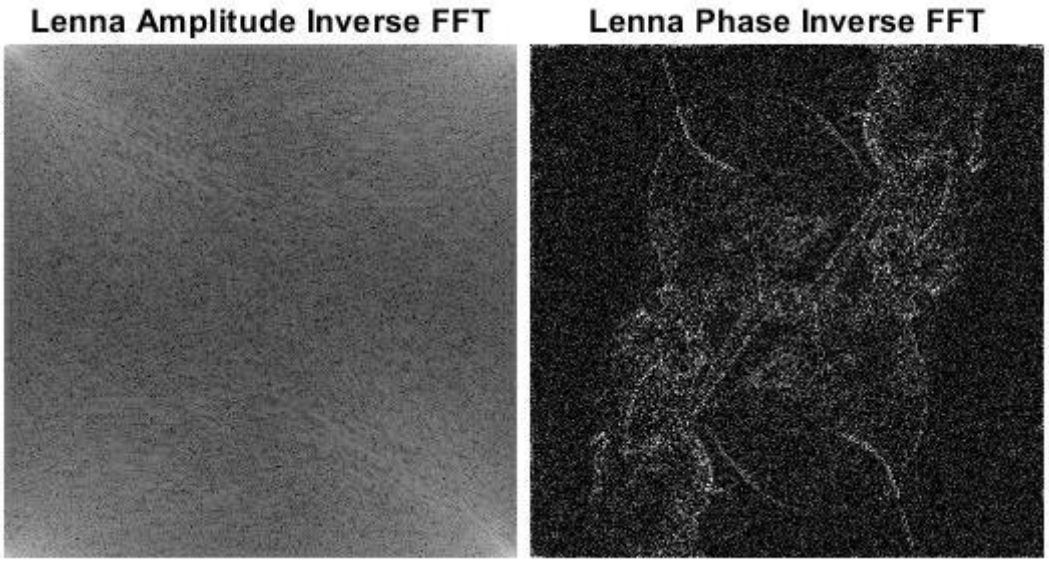
1.5.1 นำค่า Amplitude ในข้อ 1.1 มาทำกระบวนการ IFFT นั่นคือ IFFTShift และ IFFT ตามลำดับ ก็จะได้รูปของ IFFT โดยผลลัพธ์ จะออกมาเป็นข้อมูลความเข้มของแสงในรูป

1.5.2 นำค่า Phase ในข้อ 1.1 มาทำเช่นเดียวกับข้อ 1.5.1 เลย แต่ในตอนท้ายก่อนได้ผลลัพธ์ต้องทำการหา abs ก่อน โดยผลลัพธ์ที่ออกมา จะมุมของรูป เหลี่ยมอย่างชัดเจน

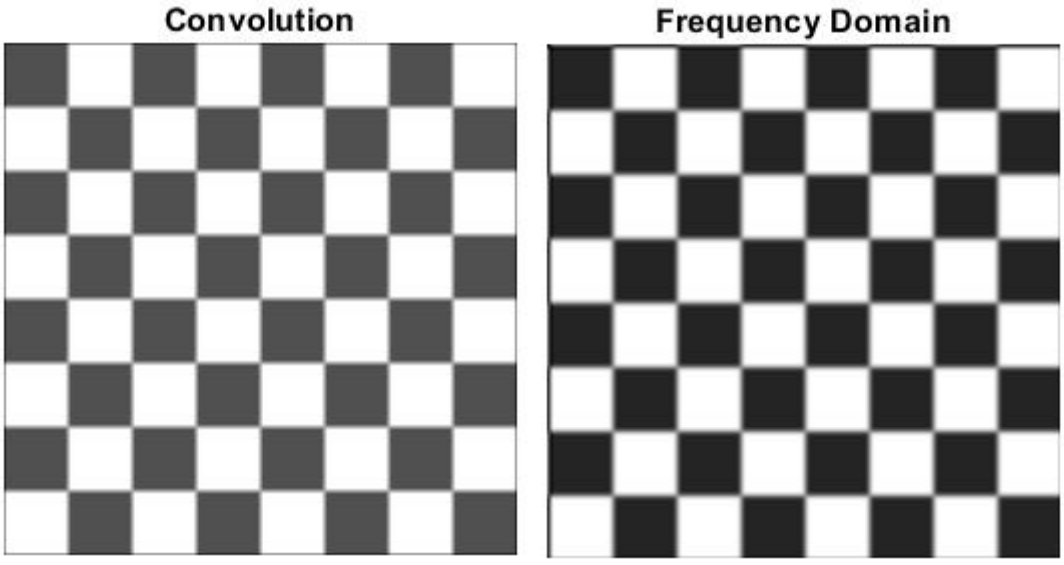




1.6 ทำเหมือนข้อ 1.5.1 และ 1.5.2 เลย เพียงเปลี่ยนรูป input เป็น Lenna โดยผลลัพธ์ก็ออกมาในกรณีเดียวกันกับข้อ 1.5 เลย นั่นคือ เป็น ค่าความเข้มแสง และ มุมของรูป input ดังรูป



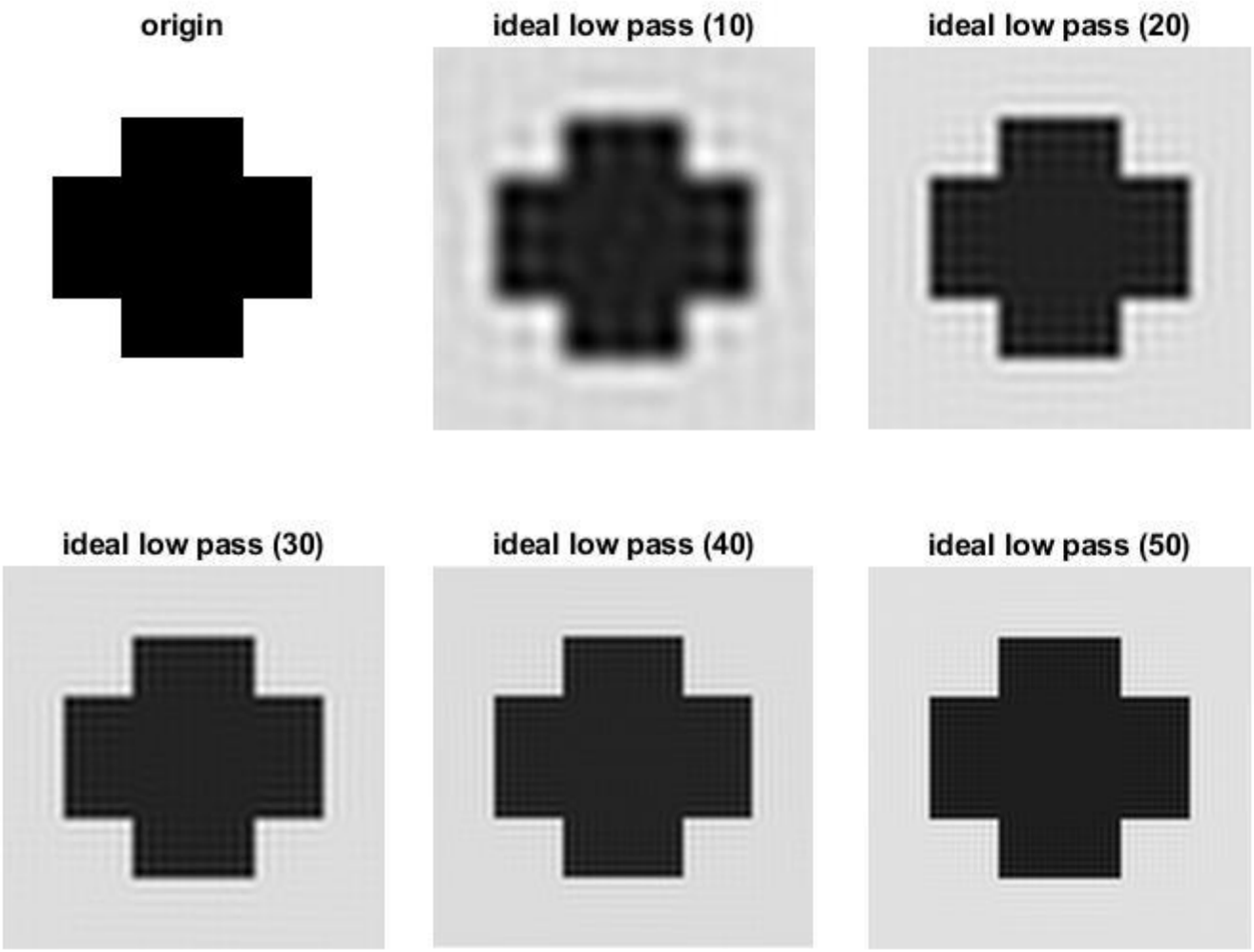
1.7 กำหนด kernel ที่มีขนาด 3x3 ขึ้นมา 1 อัน และนำไป Convolution กับรูป input ก็จะได้รูป ที่ blur ด้วย Convolution และ นำไป filter ด้วย FFT ของ kernel โดย เราจะนำ kernel ที่กำหนด มา pad ให้มีขนาดเท่ากันกับรูป input แล้วทำกระบวนการ FFT กับ kernel ที่ pad แล้ว เมื่อได้ค่ามา ให้นำค่านั้น ไปคูณ เข้ากับ ค่า FFT ของ รูปที่ blur ด้วย convolution และมาทำกระบวนการ IFFT ตามลำดับ โดย ผลลัพธ์ที่ได้ จะเห็นว่ารูปทั้ง 2 มีการเบลอ แต่ ในกรณีของ filter ด้วย frequency domain นั้น รูปที่ได้จะดูชัดเจนกว่ากรณีของ Blur ด้วย Convolution ดังรูป



2. Filter Design

2.1 Ideal low-pass filter

โดยผลลัพธ์ที่ได้นั้น เมื่อใช้ Cutoff ต่ำ Output ได้นั้นจะเกิด Ringing effect ทำให้มองเห็นภาพได้ไม่ชัด แต่เมื่อใช้ Cutoff ที่สูงขึ้น ภาพก็จะชัดขึ้นแต่ก็ยังเกิด Ring effect ขึ้นอยู่แต่ไม่มากเท่าในตอนที่ Cutoff น้อย ๆ และพื้นหลังของ output นั้นมีสีเทาไม่เป็นสีขาวตามรูป Input โดยได้ผลลัพธ์ดังรูป



Gaussian lowpass filter

โดยผลลัพธ์ที่ได้นั้น เมื่อใช้ Cutoff ต่ำ ภาพ Output ที่ได้นั้นก็จะออกมา เบลอ และยิ่ง cutoff สูง ก็ยิ่งชัดขึ้นเรื่อย ๆ และ ในการ Filter นี้ไม่เกิด Ringing effect แต่พื้นหลังของ Output จะออกเทา มากกว่า รูป Input ที่ได้ก็ ออกมาเป็นสีขาวตาม Input ด้วย

origin

Gaussian lowpass filter (10)

Gaussian lowpass filter (20)

Gaussian lowpass filter (30)

Gaussian lowpass filter (40)

Gaussian lowpass filter (50)

Butterworth lowpass filter

กำหนด n = 1 โดยผลลัพธ์ที่ได้นั้น จะเหมือนกับ 2 กรณีที่ผ่านมา ที่เมื่อ Cutoff ต่ำ ภาพที่ได้ก็จะออกมาเบลอ โดยยิ่ง Cutoff สูงภาพก็ยิ่งชัดขึ้น แต่ในกรณีของ Butterworth นั้น จะไม่เกิด Ringing effect โดยได้ผลลัพธ์ดังรูป

origin

Butterworth lowpass filter (10)

Butterworth lowpass filter (20)

Butterworth lowpass filter (30)

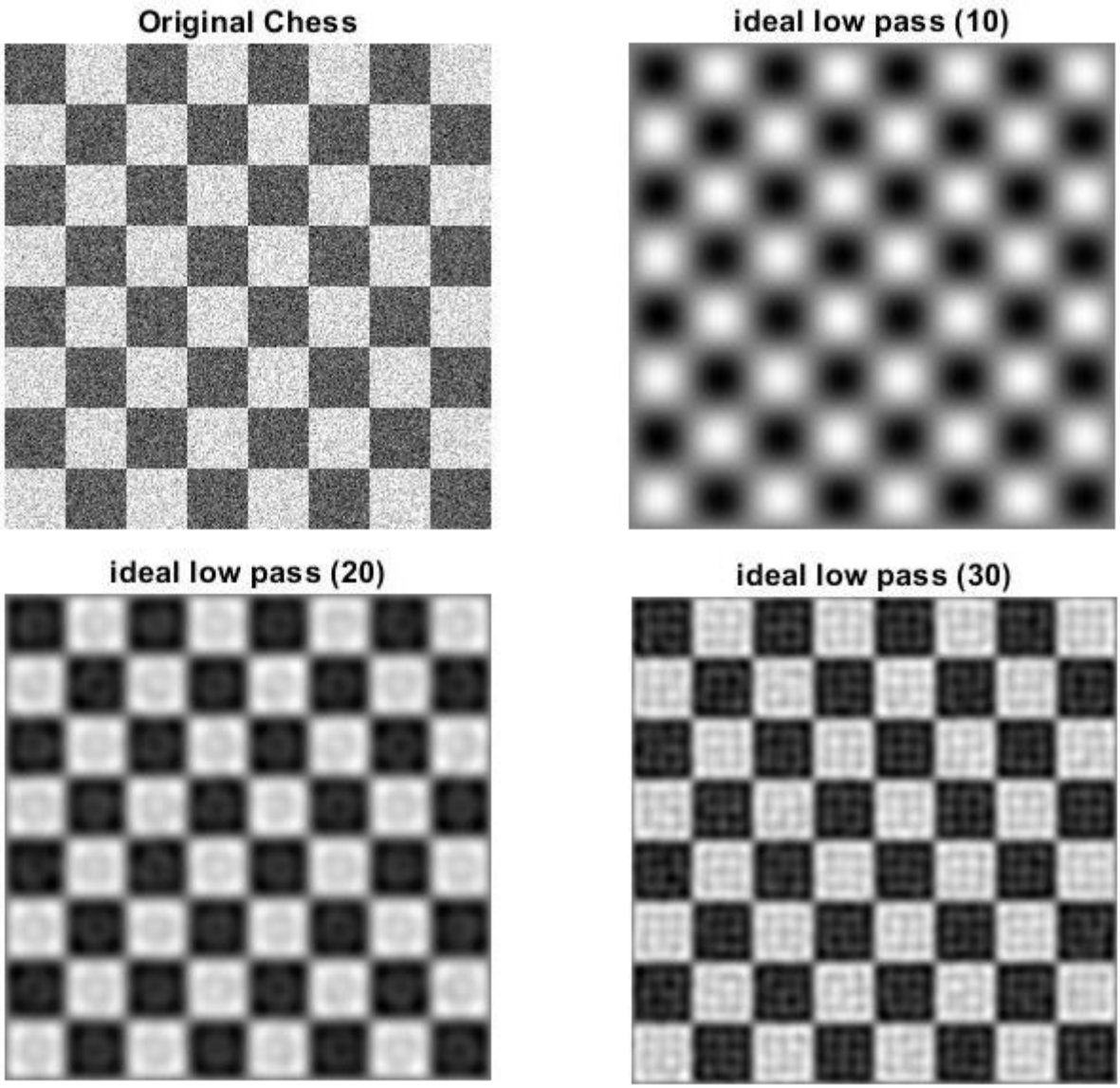
Butterworth lowpass filter (40)

Butterworth lowpass filter (50)

2.2 ในการจะหา *RMS* เราจะต้อง นำค่า *single* ของ รูปต้นฉบับที่ไม่มี *noise* มาลบด้วย *single* ของรูปที่มี *noise* และนำไปหา *sum* ของค่า *error* ที่หารด้วย *numel* และนำค่าที่ได้ไปหา *sqrt* ก็จะได้ค่า *RMS* ที่ต้องการ โดยจะได้ผลลัพธ์ดังรูป

กรณี *Chess.pgm*

Ideal low-pass filter



ideal low pass (40)

ideal low pass (50)

RMS

Cutoff 10 = 35.1179

Cutoff 20 = 22.1757

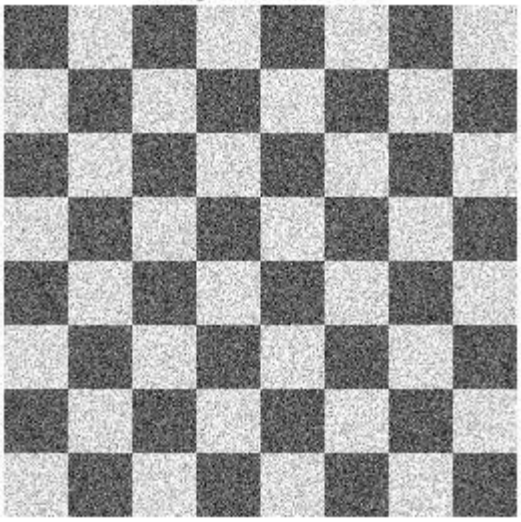
Cutoff 30 = 19.1067

Cutoff 40 = 17.5685

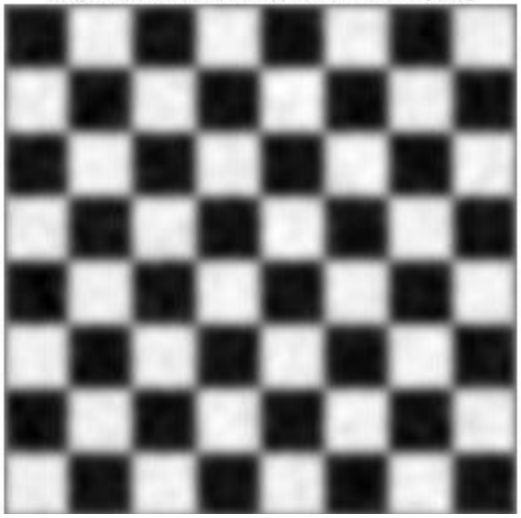
Cutoff 50 = 16.7716

Gaussian lowpass filter

Original Chess



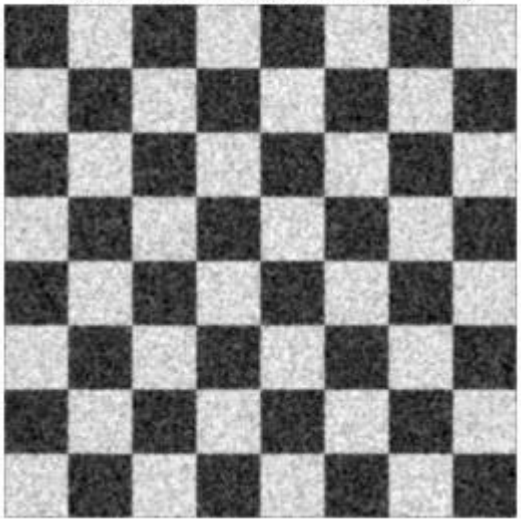
Gaussian lowpass filter (10)



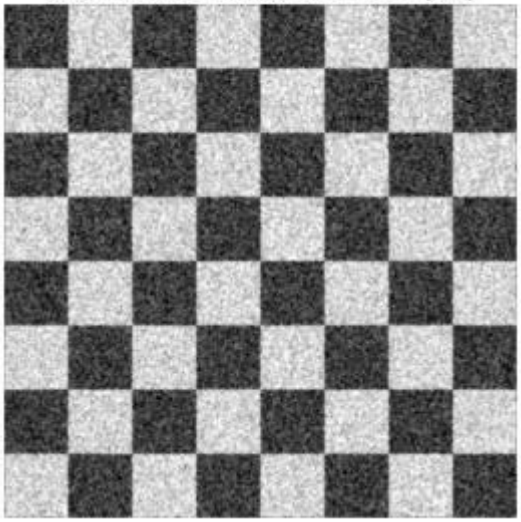
Gaussian lowpass filter (20)

Gaussian lowpass filter (30)

Gaussian lowpass filter (40)



Gaussian lowpass filter (50)



RMS

Cutoff 10 = 24.6600

Cutoff 20 = 17.4969

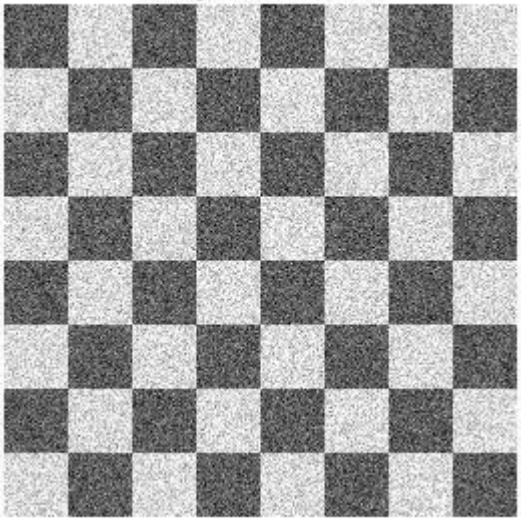
Cutoff 30 = 14.7397

Cutoff 40 = 13.8690

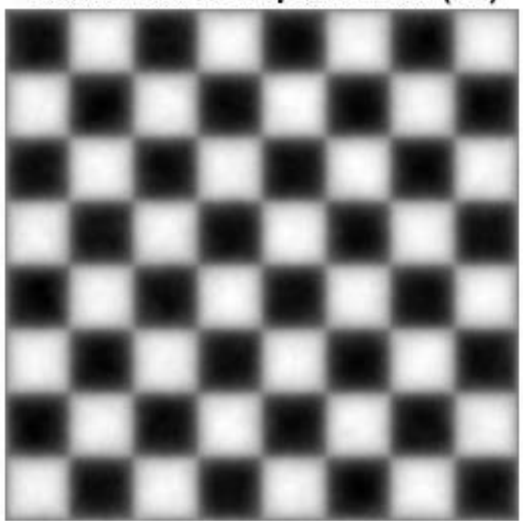
Cutoff 50 = 14.2413

Butterworth lowpass filter

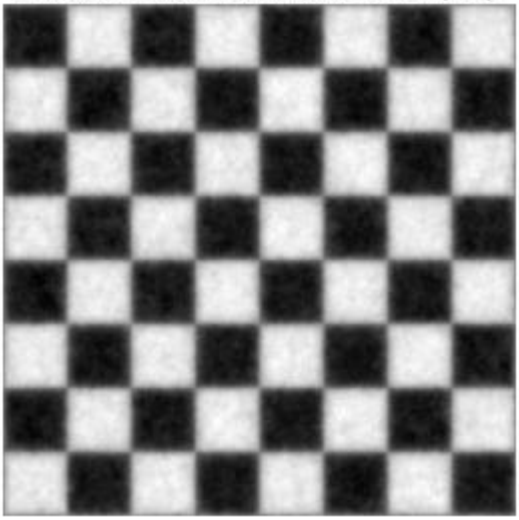
Original Chess



Butterworth lowpass filter (10)

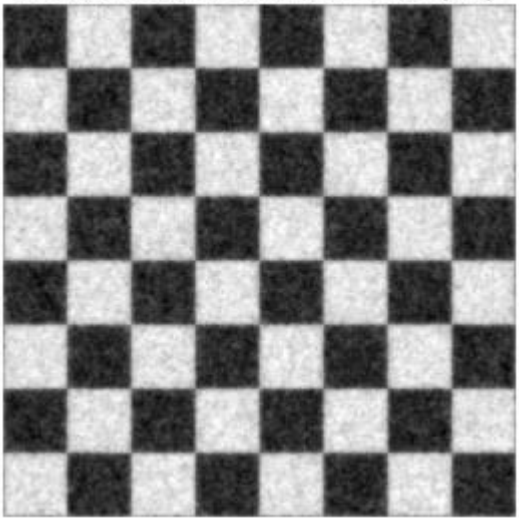


Butterworth lowpass filter (20)

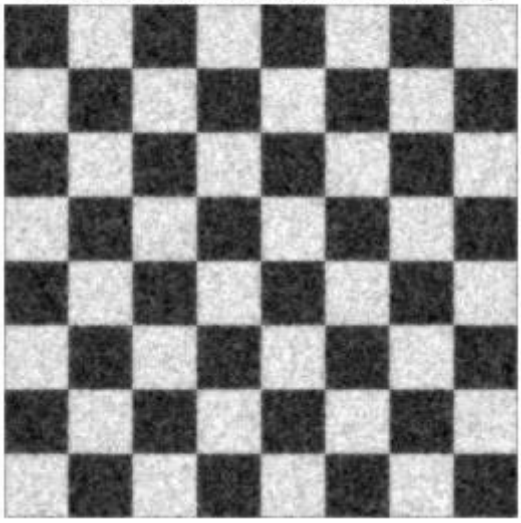


Butterworth lowpass filter (30)

Butterworth lowpass filter (40)



Butterworth lowpass filter (50)



RMS

Cutoff 10 = 30.5152

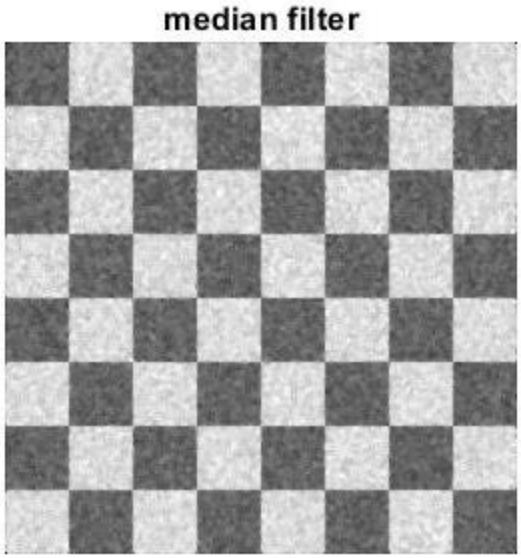
Cutoff 20 = 21.3911

Cutoff 30 = 17.5166

Cutoff 40 = 15.4820

Cutoff 50 = 14.4482

Median filter



RMS

Median Filter = 12.577671

กรณี Lenna.pgm

Ideal low-pass filter

Original Lenna

ideal low pass (10)

ideal low pass (20)

ideal low pass (30)

ideal low pass (40)

ideal low pass (50)

RMS

Cutoff 10 = 23.3992

Cutoff 20 = 16.9965

Cutoff 30 = 13.9135

Cutoff 40 = 12.8150

Cutoff 50 = 12.4744

Gaussian lowpass filter

Original Lenna

Gaussian lowpass filter (10)

Gaussian lowpass filter (20)

Gaussian lowpass filter (30)

Gaussian lowpass filter (40)

Gaussian lowpass filter (50)

RMS

Cutoff 10 = 17.3594

Cutoff 20 = 12.2103

Cutoff 30 = 11.0236

Cutoff 40 = 11.6615

Cutoff 50 = 13.1515

Butterworth lowpass filter

Original Lenna

Butterworth lowpass filter (10)

Butterworth lowpass filter (20)

Butterworth lowpass filter (30)

Butterworth lowpass filter (40)

Butterworth lowpass filter (50)

RMS

Cutoff 10 = 21.1669

Cutoff 20 = 14.9381

Cutoff 30 = 12.3918

Cutoff 40 = 11.4126

Cutoff 50 = 11.3185

Median filter

RMS

Median Filter = 12.95547

ภาคผนวก

```
% 1.1 %
ima = imread('Cross.pgm');
padimage = padarray(ima,[28 28],'both');
imagefft = fft2(double(padimage));
shiftedfft = fftshift(imagefft);
amp = log(abs(shiftedfft));
figure;
imshow(amp,[]);
colormap gray
title('Amplitude');
phase = angle(shiftedfft);
figure;
imshow(phase,[]);
colormap gray
title('Phase spectra');

% 1.2 %
x = -128:127;
y = -128:127;
[X,Y] = meshgrid(x,y);
complex_number = exp(-2j*pi*(((20*X)/256)+((30*Y)/256)));
new_phase = complex_number.*phase;
ifftshifted = ifftshift(new_phase);
img_ifft = ifft2(ifftshifted);
img_ifft = abs(img_ifft);
figure;
imshow(img_ifft);
colormap gray
title('inverse fourier transform');

% 1.3 %
rotate_img = imrotate(ima,30,'bilinear','crop');
rotate_padimage = padarray(rotate_img,[28 28],255,'both');
rotate_imagefft = fft2(double(rotate_padimage));
rotate_shiftedfft = fftshift(rotate_imagefft);
rotate_amp = log(abs(rotate_shiftedfft));
figure;
imshow(rotate_padimage);
```

```
colormap gray
title('Image Rotate');
figure;
imshow(rotate_amp,[]);
colormap gray
title('Amplitude Rotate');
rotate_phase = angle(rotate_shiftedfft);
figure;
imshow(rotate_phase,[]);
colormap gray
title('Phase spectra Rotate');

% 1.4 %
ima_down = ima(1:2:end,1:2:end);
padimage_down = padarray(ima_down,[14 14],'both');
imagefft_down = fft2(double(padimage_down));
shiftedfft_down = fftshift(imagefft_down);
amp_down = log(abs(shiftedfft_down));
figure;
imshow(amp_down,[]);
colormap gray
title('Amplitude Down');
phase_down = angle(shiftedfft_down);
figure;
imshow(phase_down,[]);
colormap gray
title('Phase spectra Down');

% 1.5.1 %
inverse_amp = ifftshift(amp);
ifft_amp = ifft2(inverse_amp);
figure;
imshow(inverse_amp,[]);
colormap gray
title('Amplitude Inverse FFT');

% 1.5.2 %
inverse_phase = ifftshift(phase);
inverse_phase = ifft2(inverse_phase);
inverse_phase = abs(inverse_phase);
figure;
imshow(inverse_phase,[]);
colormap gray
```

```
title('Phase Inverse FFT');

% 1.6 %
ima_lenna = imread('Lenna.pgm');
imagefft_lenna = fft2(double(ima_lenna));
shiftedfft_lenna = fftshift(imagefft_lenna);
amp_lenna = log(abs(shiftedfft_lenna));
phase_lenna = angle(shiftedfft_lenna);

inverse_amp_lenna = ifftshift(amp_lenna);
ifft_amp_lenna = ifft2(inverse_amp_lenna);
figure;
imshow(inverse_amp_lenna,[]);
colormap gray
title('Lenna Amplitude Inverse FFT');

inverse_phase_lenna = ifftshift(phase_lenna);
inverse_phase_lenna = ifft2(inverse_phase_lenna);
inverse_phase_lenna = abs(inverse_phase_lenna);
figure;
imshow(inverse_phase_lenna,[]);
colormap gray
title('Lenna Phase Inverse FFT');

% 1.7 %
ima_chess = imread('Chess.pgm');
ima_chess = padarray(ima_chess,[1 1],'both');
ima_chess_new = ima_chess(2:end-1,2:end-1);
[X,Y] = size(ima_chess_new);
kernel = ones(3,3);
kernel = kernel./9;

for i = 1:X
    for j = 1:Y
        avg = 0;
        for k = 1:3
            for l = 1:3
                avg = avg + ima_chess(i+k-1,j+l-
1).*kernel(k,l);
            end
        end
        ima_chess_new(i,j) = double(avg);
    end
```

```
end

ima_chess_fft = fft2(double(ima_chess_new));
shifted_ima_chess_fft = fftshift(ima_chess_fft);


pad_kernel = padarray(kernel,[253 253],'post');
kernel_fft = fft2(double(pad_kernel));
shifted_kernel_fft = fftshift(kernel_fft);

ima_chess_fillter =
shifted_kernel_fft.*shifted_ima_chess_fft;
ima_blur_ifft = ifftshift(ima_chess_fillter);
ima_blur_ifft = ifft2(ima_blur_ifft);
ima_blur_ifft = abs(ima_blur_ifft);

figure;
imshow(ima_chess_new,[]);
colormap gray
title('Convolution');

figure;
imshow(ima_blur_ifft,[]);
colormap gray
title('Frequency Domain');



  % 2.1 %
  img = imread('Cross.pgm');
  [X,Y] = size(img);
  img_fft = fft2(double(img));
  figure
  imshow(img,[]);
  title('origin');
  u = 1 : X;
  v = 1 : Y;

  currentx = find(u>X/2);
  u(currentx) = u(currentx)-X;

  currenty = find(v>Y/2);
  v(currenty) = v(currenty)-Y;
```

```
[U,V] = meshgrid(u,v);
D = sqrt((U.^2) + (V.^2));

% ideal low pass %
H10 = double(D <= 10);
G10 = H10.*img_fft;
ideal10 = ifft2(G10);
figure;
imshow(ideal10,[]);
title('ideal low pass (10)');

H20 = double(D <= 20);
G20 = H20.*img_fft;
ideal20 = ifft2(G20);
figure;
imshow(ideal20,[]);
title('ideal low pass (20)');

H30 = double(D <= 30);
G30 = H30.*img_fft;
ideal30 = ifft2(G30);
figure;
imshow(ideal30,[]);
title('ideal low pass (30)');

H40 = double(D <= 40);
G40 = H40.*img_fft;
ideal40 = ifft2(G40);
figure;
imshow(ideal40,[]);
title('ideal low pass (40)');

H50 = double(D <= 50);
G50 = H50.*img_fft;
ideal50 = ifft2(G50);
figure;
imshow(ideal50,[]);
title('ideal low pass (50)');

% gaussian lowpass filter %

H_glf10 = exp(-(D.^2)./(2.*10).^2);
G_glf10 = H_glf10.*img_fft;
```

```
GLF10 = ifft2(G_glf10);
figure;
imshow(GLF10,[]);
title('Gaussian lowpass filter (10)' );

H_glf20 = exp(-(D.^2)./(2.*20).^2);
G_glf20 = H_glf20.*img_fft;
GLF20 = ifft2(G_glf20);
figure;
imshow(GLF20,[]);
title('Gaussian lowpass filter (20)' );

H_glf30 = exp(-(D.^2)./(2.*30).^2);
G_glf30 = H_glf30.*img_fft;
GLF30 = ifft2(G_glf30);
figure;
imshow(GLF30,[]);
title('Gaussian lowpass filter (30)' );

H_glf40 = exp(-(D.^2)./(2.*40).^2);
G_glf40 = H_glf40.*img_fft;
GLF40 = ifft2(G_glf40);
figure;
imshow(GLF40,[]);
title('Gaussian lowpass filter (40)' );

H_glf50 = exp(-(D.^2)./(2.*50).^2);
G_glf50 = H_glf50.*img_fft;
GLF50 = ifft2(G_glf50);
figure;
imshow(GLF50,[]);
title('Gaussian lowpass filter (50)' );

% butterworth lowpass filter %
n = 1;

H_blf10 = (1 + ((D./10).^(2*n))).^(-1);
G_blf10 = H_blf10.*img_fft;
BLF10 = ifft2(G_blf10);
figure;
imshow(BLF10,[]);
title('Butterworth lowpass filter (10)');
```

```
H_blf20 = (1 + ((D./20).^(2*n))).^(-1);
G_blf20 = H_blf20.*img_fft;
BLF20 = ifft2(G_blf20);
figure;
imshow(BLF20,[]);
title('Butterworth lowpass filter (20)');

H_blf30 = (1 + ((D./30).^(2*n))).^(-1);
G_blf30 = H_blf30.*img_fft;
BLF30 = ifft2(G_blf30);
figure;
imshow(BLF30,[]);
title('Butterworth lowpass filter (30)');

H_blf40 = (1 + ((D./40).^(2*n))).^(-1);
G_blf40 = H_blf40.*img_fft;
BLF40 = ifft2(G_blf40);
figure;
imshow(BLF40,[]);
title('Butterworth lowpass filter (40)');

H_blf50 = (1 + ((D./50).^(2*n))).^(-1);
G_blf50 = H_blf50.*img_fft;
BLF50 = ifft2(G_blf50);
figure;
imshow(BLF50,[]);
title('Butterworth lowpass filter (50)');


% 2.2 %
%img_noise = imread('Lenna_noise.pgm');
img_ori = imread('Lenna.pgm');
img_noise = imread('Chess_noise.pgm');
%img_ori = imread('Chess.pgm');
[X,Y] = size(img_noise);
img_fft = fft2(double(img_noise));

u = 1 : X;
v = 1 : Y;

currentx = find(u>X/2);
u(currentx) = u(currentx)-X;
```

```
currenty = find(v>Y/2);
v(currenty) = v(currenty)-Y;

[U,V] = meshgrid(u,v);
D = sqrt((U.^2) + (V.^2));

% ideal low pass %
H10 = double(D <= 10);
G10 = H10.*img_fft;
ideal10 = ifft2(G10);
figure;
imshow(ideal10,[]);
title('ideal low pass (10)');

H20 = double(D <= 20);
G20 = H20.*img_fft;
ideal20 = ifft2(G20);
figure;
imshow(ideal20,[]);
title('ideal low pass (20)');

H30 = double(D <= 30);
G30 = H30.*img_fft;
ideal30 = ifft2(G30);
figure;
imshow(ideal30,[]);
title('ideal low pass (30)');

H40 = double(D <= 40);
G40 = H40.*img_fft;
ideal40 = ifft2(G40);
figure;
imshow(ideal40,[]);
title('ideal low pass (40)');

H50 = double(D <= 50);
G50 = H50.*img_fft;
ideal50 = ifft2(G50);
figure;
imshow(ideal50,[]);
title('ideal low pass (50)');
```

```
% gaussian lowpass filter %

  H_glf10 = exp(-(D.^2)./(2.*10).^2);
  G_glf10 = H_glf10.*img_fft;
  GLF10 = ifft2(G_glf10);
  figure;
  imshow(GLF10,[]);
  title('Gaussian lowpass filter (10)' );

  H_glf20 = exp(-(D.^2)./(2.*20).^2);
  G_glf20 = H_glf20.*img_fft;
  GLF20 = ifft2(G_glf20);
  figure;
  imshow(GLF20,[]);
  title('Gaussian lowpass filter (20)' );

  H_glf30 = exp(-(D.^2)./(2.*30).^2);
  G_glf30 = H_glf30.*img_fft;
  GLF30 = ifft2(G_glf30);
  figure;
  imshow(GLF30,[]);
  title('Gaussian lowpass filter (30)' );

  H_glf40 = exp(-(D.^2)./(2.*40).^2);
  G_glf40 = H_glf40.*img_fft;
  GLF40 = ifft2(G_glf40);
  figure;
  imshow(GLF40,[]);
  title('Gaussian lowpass filter (40)' );

  H_glf50 = exp(-(D.^2)./(2.*50).^2);
  G_glf50 = H_glf50.*img_fft;
  GLF50 = ifft2(G_glf50);
  figure;
  imshow(GLF50,[]);
  title('Gaussian lowpass filter (50)' );

  % butterworth lowpass filter %
  n = 1;

  H_blf10 = (1 + ((D./10).^(2*n))).^(-1);
  G_blf10 = H_blf10.*img_fft;
  BLF10 = ifft2(G_blf10);
```

```
figure;
imshow(BLF10,[]);
title('Butterworth lowpass filter (10)');

H_blf20 = (1 + ((D./20).^(2*n))).^(-1);
G_blf20 = H_blf20.*img_fft;
BLF20 = ifft2(G_blf20);
figure;
imshow(BLF20,[]);
title('Butterworth lowpass filter (20)');

H_blf30 = (1 + ((D./30).^(2*n))).^(-1);
G_blf30 = H_blf30.*img_fft;
BLF30 = ifft2(G_blf30);
figure;
imshow(BLF30,[]);
title('Butterworth lowpass filter (30)');

H_blf40 = (1 + ((D./40).^(2*n))).^(-1);
G_blf40 = H_blf40.*img_fft;
BLF40 = ifft2(G_blf40);
figure;
imshow(BLF40,[]);
title('Butterworth lowpass filter (40)');

H_blf50 = (1 + ((D./50).^(2*n))).^(-1);
G_blf50 = H_blf50.*img_fft;
BLF50 = ifft2(G_blf50);
figure;
imshow(BLF50,[]);
title('Butterworth lowpass filter (50)');

  % RMS ideal low pass %
  ideal_er10 = (single(img_ori) - single(ideal10)).^2;
  ideal_mean10 = sum(ideal_er10(:)) / numel(img_ori);
  ideal_rmsError10 = sqrt(ideal_mean10);

  ideal_er20 = (single(img_ori) - single(ideal20)).^2;
  ideal_mean20 = sum(ideal_er20(:)) / numel(img_ori);
  ideal_rmsError20 = sqrt(ideal_mean20);

  ideal_er30 = (single(img_ori) - single(ideal30)).^2;
  ideal_mean30 = sum(ideal_er30(:)) / numel(img_ori);
```

```
ideal_rmsError30 = sqrt(ideal_mean30);

ideal_er40 = (single(img_ori) - single(ideal40)).^2;
ideal_mean40 = sum(ideal_er40(:)) / numel(img_ori);
ideal_rmsError40 = sqrt(ideal_mean40);

ideal_er50 = (single(img_ori) - single(ideal50)).^2;
ideal_mean50 = sum(ideal_er50(:)) / numel(img_ori);
ideal_rmsError50 = sqrt(ideal_mean50);

% RMS gaussian lowpass filter %

GLF_er10 = (single(img_ori) - single(GLF10)).^2;
GLF_mean10 = sum(GLF_er10(:)) / numel(img_ori);
GLF_rmsError10 = sqrt(GLF_mean10);

GLF_er20 = (single(img_ori) - single(GLF20)).^2;
GLF_mean20 = sum(GLF_er20(:)) / numel(img_ori);
GLF_rmsError20 = sqrt(GLF_mean20);

GLF_er30 = (single(img_ori) - single(GLF30)).^2;
GLF_mean30 = sum(GLF_er30(:)) / numel(img_ori);
GLF_rmsError30 = sqrt(GLF_mean30);

GLF_er40 = (single(img_ori) - single(GLF40)).^2;
GLF_mean40 = sum(GLF_er40(:)) / numel(img_ori);
GLF_rmsError40 = sqrt(GLF_mean40);

GLF_er50 = (single(img_ori) - single(GLF50)).^2;
GLF_mean50 = sum(GLF_er50(:)) / numel(img_ori);
GLF_rmsError50 = sqrt(GLF_mean50);

% RMS butterworth lowpass filter %

BLF_er10 = (single(img_ori) - single(BLF10)).^2;
BLF_mean10 = sum(BLF_er10(:)) / numel(img_ori);
BLF_rmsError10 = sqrt(BLF_mean10);

BLF_er20 = (single(img_ori) - single(BLF20)).^2;
BLF_mean20 = sum(BLF_er20(:)) / numel(img_ori);
BLF_rmsError20 = sqrt(BLF_mean20);

BLF_er30 = (single(img_ori) - single(BLF30)).^2;
BLF_mean30 = sum(BLF_er30(:)) / numel(img_ori);
```

```
    BLF_rmsError30 = sqrt(BLF_mean30);

    BLF_er40 = (single(img_ori) - single(BLF40)).^2;
    BLF_mean40 = sum(BLF_er40(:)) / numel(img_ori);
    BLF_rmsError40 = sqrt(BLF_mean40);

    BLF_er50 = (single(img_ori) - single(BLF50)).^2;
    BLF_mean50 = sum(BLF_er50(:)) / numel(img_ori);
    BLF_rmsError50 = sqrt(BLF_mean50);


   % Median filter %

  img_noise_pad = padarray(img_noise , [1 1] , 'both');
  [X,Y] = size(img_noise);
  img_noise_new = img_noise_pad(2:end-1,2:end-1);
  arr = ones(9,1);

  for i = 1:X
      for j = 1:Y
          current = 1;
          for m = 1:3
              for n = 1:3
                  arr(current) = img_noise_pad(i+m-1,j+n-
1);
                  current = current + 1;
              end
          end
          MF(i,j) = median(arr);
      end
  end

  figure;
  imshow(MF,[]);
  title('median filter');
 % RMS Median filter %

    MF_er = (single(img_ori) - single(MF)).^2;
    MF_mean = sum(MF_er(:)) / numel(img_ori);
    MF_rmsError = sqrt(MF_mean);
```