

Laboratory of Image Processing

Image Processing in Fourier Domain

Pier Luigi Mazzeo
pierluigi.mazzeo@cnr.it

The DFT

This is a brief review of the Fourier transform. An in-depth discussion of the Fourier transform is best left to your class instructor.

The general idea is that the image ($f(x,y)$ of size $M \times N$) will be represented in the frequency domain ($F(u,v)$). The equation for the two-dimensional discrete Fourier transform (DFT) is:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-i2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

The concept behind the Fourier transform is that any waveform can be constructed using a sum of sine and cosine waves of different frequencies. The exponential in the above formula can be expanded into sines and cosines with the variables u and v determining these frequencies. The inverse of the above discrete Fourier transform is given by the following equation:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{i2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

The DFT

Thus, if we have $F(u,v)$, we can obtain the corresponding image ($f(x,y)$) using the inverse, discrete Fourier transform.

Things to note about the discrete Fourier transform are the following:

- the value of the transform at the origin of the frequency domain, at $F(0,0)$, is called the dc component
 - $F(0,0)$ is equal to MN times the average value of $f(x,y)$
 - o in MATLAB, $F(0,0)$ is actually $F(1,1)$ because array indices in MATLAB start at 1 rather than 0
- the values of the Fourier transform are complex, meaning they have real and imaginary parts. The imaginary parts are represented by i , which is defined solely by the property that its square is -1 , ie:
$$i^2 = -1$$
- we visually analyze a Fourier transform by computing a Fourier spectrum (the magnitude of $F(u,v)$) and display it as an image.
 - the Fourier spectrum is symmetric about the origin
- the fast Fourier transform (FFT) is a fast algorithm for computing the discrete Fourier transform.

The DFT

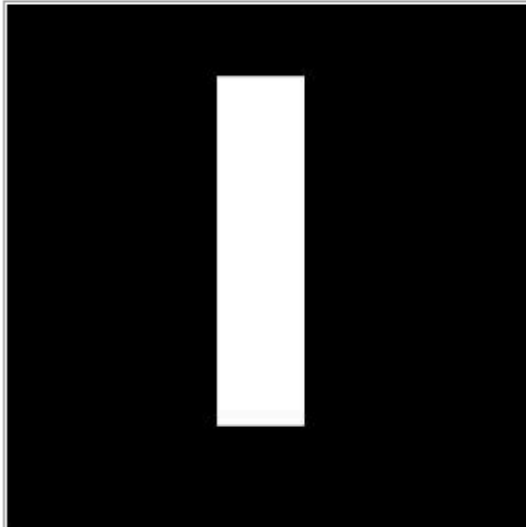
MATLAB has three functions to compute the DFT:

1. `fft` -for one dimension (useful for audio)
2. `fft2` -for two dimensions (useful for images)
3. `fftn` -for n dimensions

MATLAB has three related functions that compute the inverse DFT:

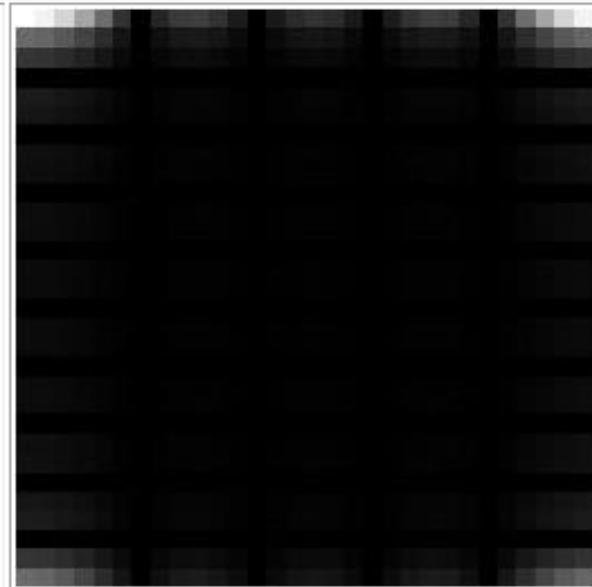
1. `ifft` -for one dimension (useful for audio)
2. `ifft2` -for two dimensions (useful for images)
3. `ifftn` -for n dimensions

How to display a Fourier Spectrum using MATLAB

MATLAB code	Image Produced
<pre>%Create a black 30x30 image f=zeros(30,30); %With a white rectangle in it. f(5:24,13:17)=1; imshow(f,'InitialMagnification','fit')</pre>	 The image produced is a 30x30 pixel grayscale image. It features a solid black background. In the center of the image, there is a white vertical rectangle. The rectangle is 15 pixels wide (from column 13 to 17) and 20 pixels high (from row 5 to 24). The image is displayed using the 'fit' option, meaning it is scaled to fit the window.

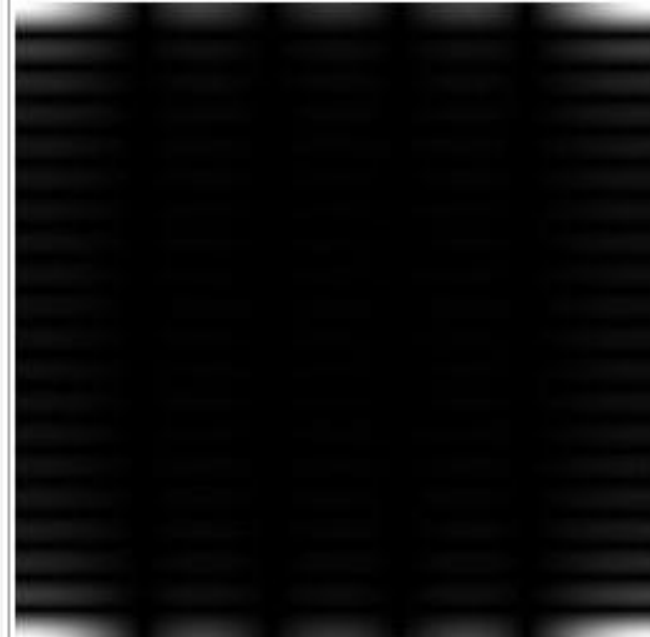
How to display a Fourier Spectrum using MATLAB

```
%Calculate the DFT.  
F=fft2(f);  
  
%There are real and imaginary parts to F.  
%Use the abs function to compute the  
magnitude  
%of the combined components.  
F2=abs(F);  
  
figure, imshow(F2,[],  
'InitialMagnification','fit')
```



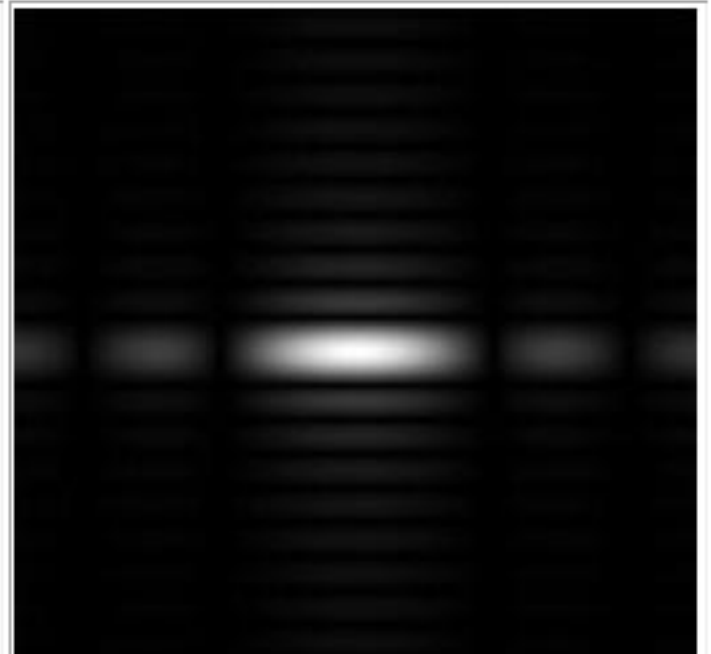
How to display a Fourier Spectrum using MATLAB

```
%To create a finer sampling of the Fourier  
transform,  
%you can add zero padding to f when  
computing its DFT  
%Also note that we use a power of 2, 2^256  
%This is because the FFT -Fast Fourier  
Transform -  
%is fastest when the image size has many  
factors.  
F=fft2(f, 256, 256);  
  
F2=abs(F);  
figure, imshow(F2, [])
```



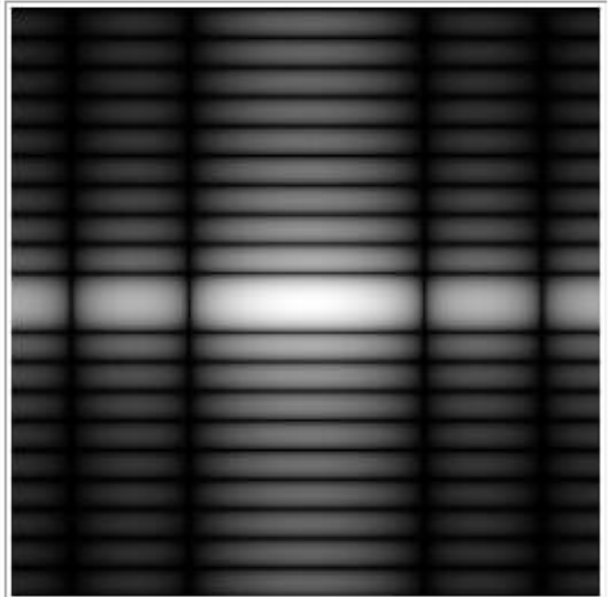
How to display a Fourier Spectrum using MATLAB

```
%The zero-frequency coefficient is  
displayed in the  
%upper left hand corner. To display it in  
the center,  
%you can use the function fftshift.  
F2=fftshift(F);  
  
F2=abs(F2);  
figure,imshow(F2,[])
```



How to display a Fourier Spectrum using MATLAB

```
%In Fourier transforms, high peaks are so  
high they  
%hide details. Reduce contrast with the log  
function.  
F2=log(1+F2);  
  
figure,imshow(F2,[])
```



How to display a Fourier Spectrum using MATLAB

To get the results shown in the last image of the table, you can also combine MATLAB calls as in:

```
f=zeros(30,30);  
f(5:24,13:17)=1;  
F=fft2(f, 256,256);  
F2=fftshift(F);  
figure,imshow(log(1+abs(F2)),[])
```

Notice in these calls to `imshow`, the second argument is empty square brackets. This maps the minimum value in the image to black and the maximum value in the image to white

Frequency Domain Versions of Spatial Filters

The following convolution theorem shows an interesting relationship between the spatial domain and frequency domain:

$$f(x,y) * h(x,y) \Leftrightarrow H(u,v) F(u,v)$$

$$f(x,y) h(x,y) \Leftrightarrow H(u,v) * G(u,v)$$

where the symbol "*" indicates convolution of the two functions. The important thing to extract out of this is that the multiplication of two Fourier transforms corresponds to the convolution of the associated functions in the spatial domain.

This means we can perform linear spatial filters as a simple component-wise multiply in the frequency domain.

This suggests that we could use Fourier transforms to speed up spatial filters. This only works for large images that are correctly padded, where multiple transformations are applied in the frequency domain before moving back to the spatial domain.

When applying Fourier transforms padding is very important. Note that, because images are infinitely tiled in the frequency domain, filtering produces wraparound artefacts if you don't zero pad the image to a larger size.

The `paddedsiz` function below calculates a correct padding size to avoid this problem. The `paddedsiz` function can also help optimize the performance of the DFT by providing power of 2 padding sizes. See `paddedsiz` function

```
function PQ = paddedsiz(AB, CD, PARAM)
%PADDEDSIZE Computes padded sizes useful for FFT-based filtering.
%   PQ = PADDEDSIZE(AB), where AB is a two-element size vector,
%   computes the two-element size vector PQ = 2*AB.
%
%   PQ = PADDEDSIZE(AB, 'PWR2') computes the vector PQ such that
%   PQ(1) = PQ(2) = 2^nextpow2(2*m), where m is MAX(AB).
%
%   PQ = PADDEDSIZE(AB, CD), where AB and CD are two-element size
%   vectors, computes the two-element size vector PQ. The elements
%   of PQ are the smallest even integers greater than or equal to
%   AB + CD - 1.
%
%   PQ = PADDEDSIZE(AB, CD, 'PWR2') computes the vector PQ such that
%   PQ(1) = PQ(2) = 2^nextpow2(2*m), where m is MAX([AB CD]).
```

```

if nargin == 1
    PQ = 2*AB;
elseif nargin == 2 & ~ischar(CD)
    PQ = AB + CD - 1;
    PQ = 2 * ceil(PQ / 2);
elseif nargin == 2
    m = max(AB); % Maximum dimension.

    % Find power-of-2 at least twice m.
    P = 2^nextpow2(2*m);
    PQ = [P, P];
elseif nargin == 3
    m = max([AB CD]); %Maximum dimension.
    P = 2^nextpow2(2*m);
    PQ = [P, P];
else
    error('Wrong number of inputs.')
end

```

Basic Steps in DFT Filtering

1. Obtain the padding parameters using function `addedsized`:

```
PQ=paddedsized(size(f));
```

2. Obtain the Fourier transform of the image with padding:

```
F=fft2(f, PQ(1), PQ(2));
```

3. Generate a filter function, `H`, the same size as the image

4. Multiply the transformed image by the filter:

```
G=H.*F;
```

5. Obtain the real part of the inverse FFT of `G`:

```
g=real(ifft2(G));
```

6. Crop the top, left rectangle to the original size:

```
g=g(1:size(f, 1), 1:size(f, 2));
```

Example: Applying the Sobel Filter in the Frequency Domain



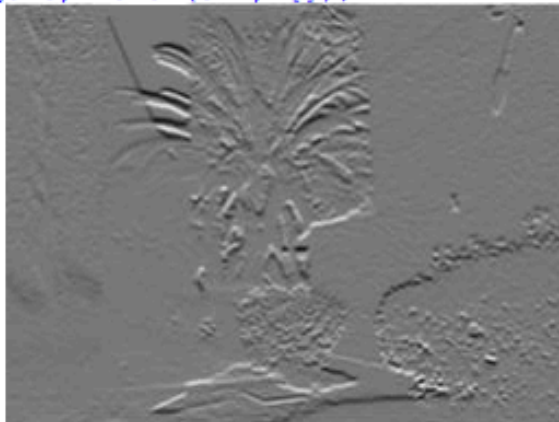
For example, let's apply the Sobel filter to the following picture in both the spatial domain and frequency domain.

Example: Applying the Sobel Filter in the Frequency Domain

Spatial Domain Filtering

```
%Create the Spacial Filtered Image  
f = imread('entry2.png');  
h = fspecial('sobel');  
sfi = imfilter(double(f),h, 0, 'conv');
```

```
%Display results (show all values)  
figure,imshow(sfi, []);
```

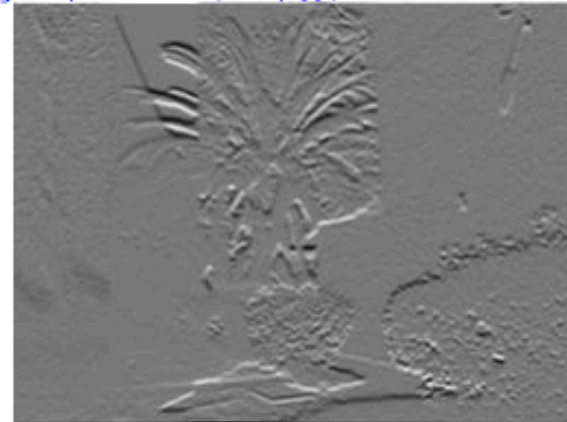


```
%The abs function gets correct  
magnitude  
%when used on complex numbers  
sfim = abs(sfi);  
figure, imshow(sfim, []);
```

Frequency Domain Filtering

```
%Create the Frequency Filtered Image  
f = imread('entry2.png');  
h = fspecial('sobel');  
PQ = paddedsize(size(f));  
F = fft2(double(f), PQ(1), PQ(2));  
H = fft2(double(h), PQ(1), PQ(2));  
F_fH = H.*F;  
ffi = ifft2(F_fH);  
ffi = ffi(2:size(f,1)+1,  
2:size(f,2)+1);
```

```
%Display results (show all values)  
figure, imshow(ffi, []);
```



```
%The abs function gets correct  
magnitude  
%when used on complex numbers  
ffim = abs(ffi);  
figure, imshow(ffim, []);
```


Example: Applying the Sobel Filter in the Frequency Domain



```
%threshold into a binary image  
figure, imshow(sfim >  
0.2*max(sfim(:)));
```



```
%threshold into a binary image  
figure, imshow(ffim >  
0.2*max(ffim(:)));
```



Frequency Domain Specific Filters

As you have already seen, based on the property that multiplying the FFT of two functions from the spatial domain produces the convolution of those functions, you can use Fourier transforms as a fast convolution on large images. Note that on small images it is faster to work in the spatial domain.

However, you can also create filters directly in the frequency domain. There are three commonly discussed filters in the frequency domain:

1. Lowpass filters, sometimes known as smoothing filters
2. Highpass filters, sometimes known as sharpening filters
3. Notch filters, sometimes known as band-stop filters

Lowpass Filters

Lowpass filters:

create a blurred (or smoothed) image

attenuate the high frequencies and leave the low frequencies of the Fourier transform relatively unchanged

Three main lowpass filters are discussed in Digital Image Processing Using MATLAB:

1. ideal lowpass filter (ILPF)
2. Butterworth lowpass filter (BLPF)
3. Gaussian lowpass filter (GLPF)

The corresponding formulas and visual representations of these filters are shown in the table below. In the formulae, **D_0** is a specified nonnegative number. **$D(u,v)$** is the distance from point **(u,v)** to the center of the filter.

```

function [U, V] = dftuv(M, N)
%DFTUV Computes meshgrid frequency matrices.
%   [U, V] = DFTUV(M, N) computes meshgrid
frequency matrices U and
%   V. U and V are useful for computing frequency-
domain filter
%   functions that can be used with DFTFILT.  U
and V are both M-by-N.

% Set up range of variables.
u = 0:(M-1);
v = 0:(N-1);

% Compute the indices for use in meshgrid
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;

% Compute the meshgrid arrays
[V, U] = meshgrid(v, u);

```

```

function H = lpfilter(type, M, N, D0, n)
%LPFILTER Computes frequency domain lowpass filters
%   H = LPFILTER(TYPE, M, N, D0, n) creates the transfer function of
%   a lowpass filter, H, of the specified TYPE and size (M-by-N). To
%   view the filter as an image or mesh plot, it should be centered
%   using H = fftshift(H).
%
%   Valid values for TYPE, D0, and n are:
%
%   'ideal'      Ideal lowpass filter with cutoff frequency D0.  n need
%                 not be supplied.  D0 must be positive
%
%   'btw'        Butterworth lowpass filter of order n, and cutoff D0.
%                 The default value for n is 1.0.  D0 must be positive.
%
%   'gaussian'   Gaussian lowpass filter with cutoff (standard deviation)
%                 D0.  n need not be supplied.  D0 must be positive.

```

```

% Use function dftuv to set up the meshgrid arrays
needed for
% computing the required distances.
[U, V] = dftuv(M, N);

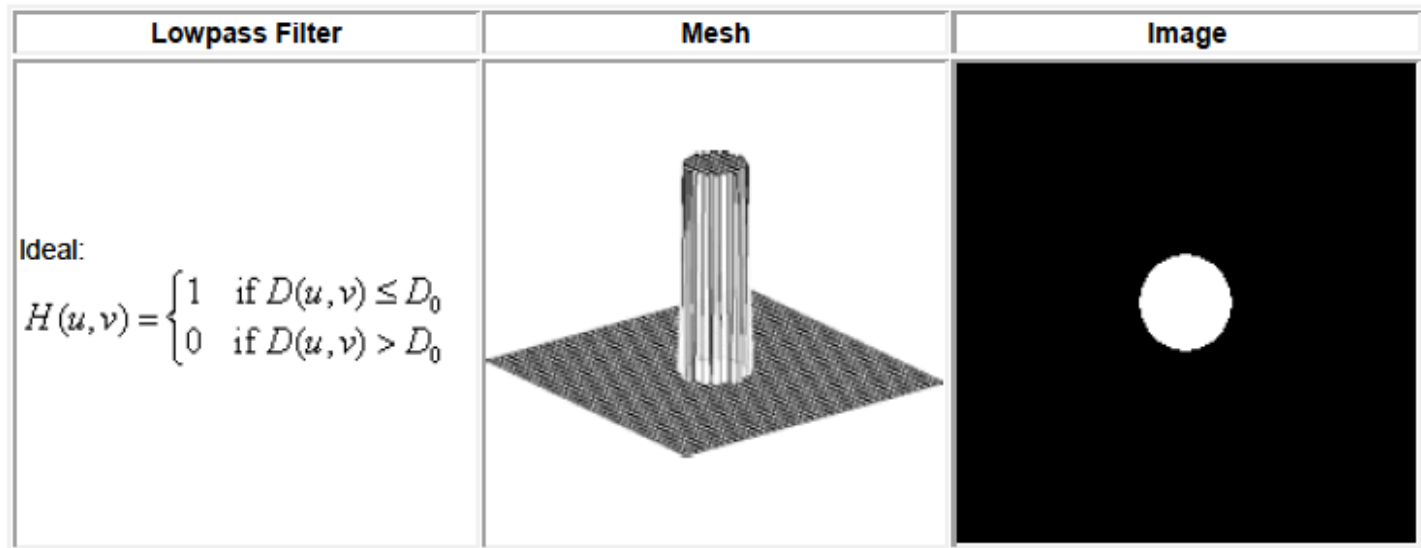
% Compute the distances D(U, V).
D = sqrt(U.^2 + V.^2);

% Begin filter computations.
switch type
case 'ideal'
    H = double(D <= D0);
case 'btw'
    if nargin == 4
        n = 1;
    end
    H = 1./(1 + (D./D0).^(2*n));
case 'gaussian'
    H = exp(-(D.^2)./(2*(D0^2)));
otherwise
    error('Unknown filter type.')
end

```

Ideal

```
HLPF_ideal=fftshift(lpfilter('ideal', 500, 500, 50));  
mesh(HLPF_ideal(1:10:500,1:10:500))  
colormap([0 0 0])  
axis off  
grid off  
axis([0 50 0 50 0 1])  
figure, imshow(HLPF_ideal)
```

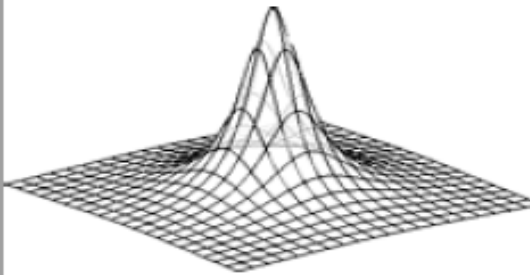


Butterworth

```
HLPF_btw=fftshift(lpfilter('btw', 500,  
500, 50));  
figure, mesh(HLPF_btw(1:10:500,1:10:500))  
colormap([0 0 0])  
axis off  
grid off  
axis([0 50 0 50 0 1])  
figure, imshow(HLPF_btw)
```

Butterworth:

$$H(u, v) = \frac{1}{1 + [D(u, v) / D_0]^{2n}}$$

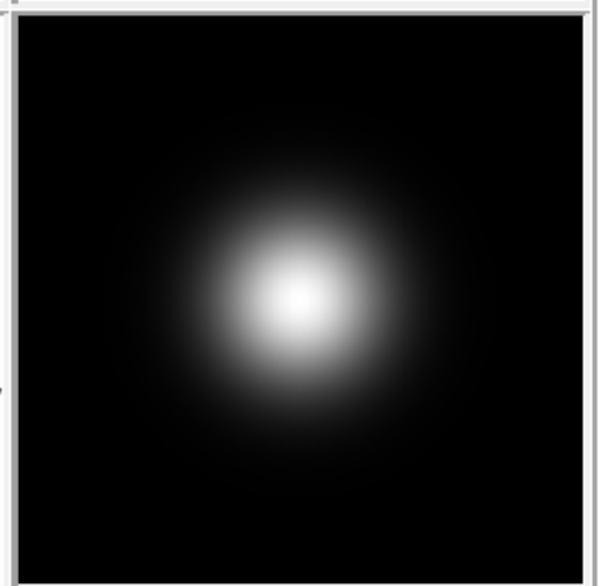
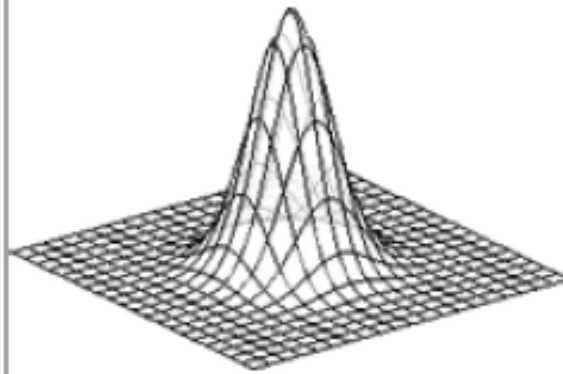


Gaussian

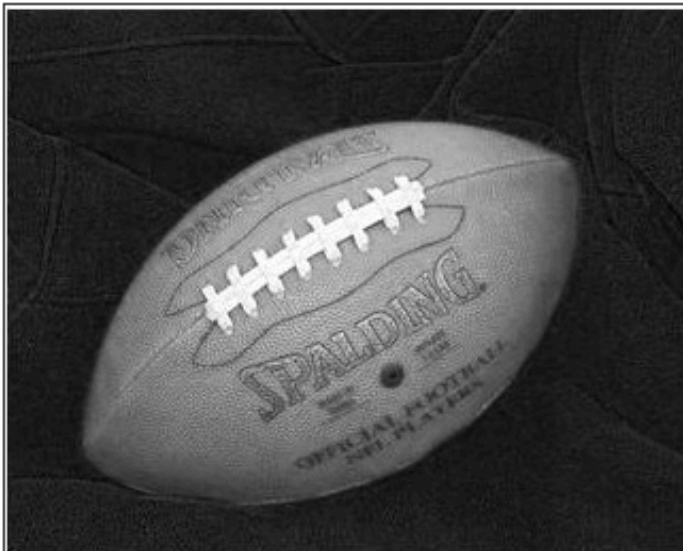
```
HLPF_gauss=fftshift(lpfilter('gaussian', 500, 500,  
50));  
figure, mesh(HLPF_gauss(1:10:500,1:10:500))  
colormap([0 0 0])  
axis off  
grid off  
axis([0 50 0 50 0 1])  
figure, imshow(HLPF_gauss)
```

Gaussian:

$$H(u, v) = e^{-D^2(u, v) / 2D_0^2}$$



Original Image



Fourier Spectrum of Image

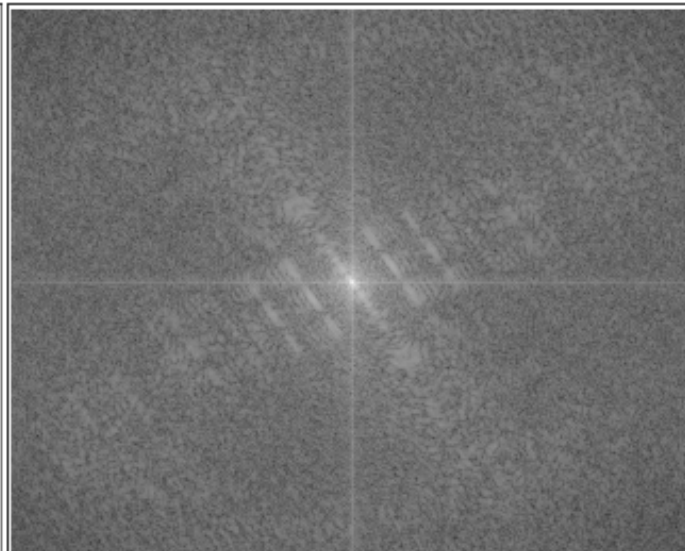
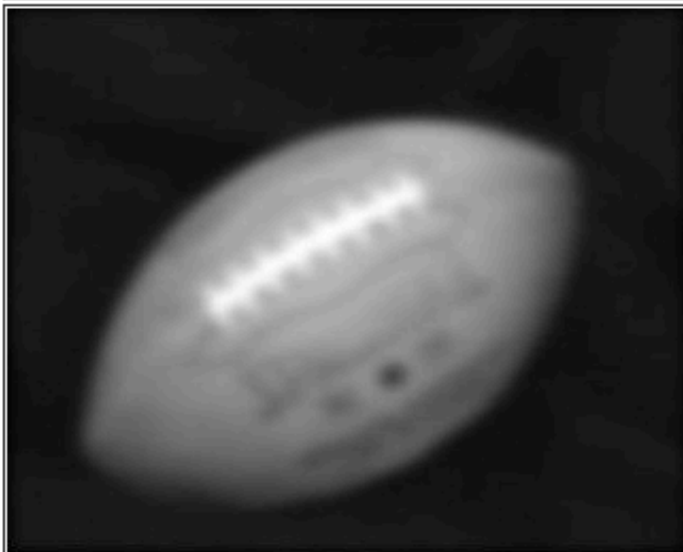
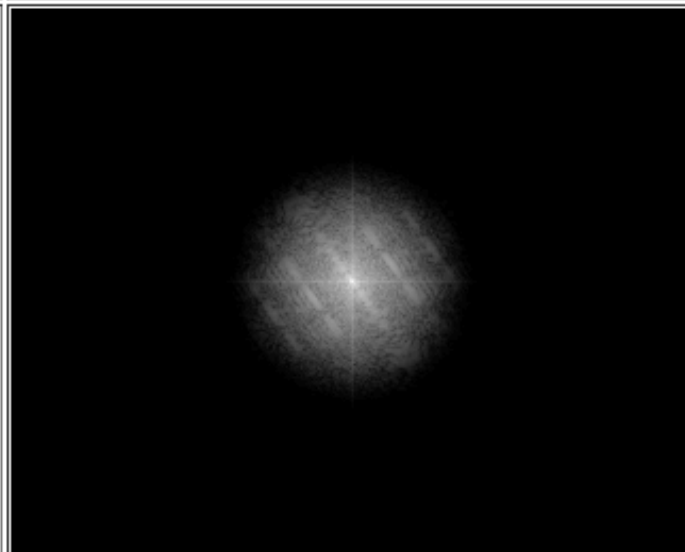


Image with Gaussian lowpass filter



Spectrum of image with Gaussian lowpass filter



```
footBall=imread('football.jpg');
%Convert to grayscale
footBall=rgb2gray(footBall);
imshow(footBall)
%Determine good padding for Fourier transform
PQ = paddedsize(size(footBall));
%Create a Gaussian Lowpass filter 5% the width of
the Fourier transform
D0 = 0.05*PQ(1);
H = lpfilter('gaussian', PQ(1), PQ(2), D0);

% Calculate the discrete Fourier transform of the
image
F=fft2(double(footBall),size(H,1),size(H,2));

% Apply the highpass filter to the Fourier spectrum
of the image
LPFS_football = H.*F;

% convert the result to the spacial domain.
LPF_football=real(ifft2(LPFS_football));
```

```
% Crop the image to undo padding
LPF_football=LPF_football(1:size(footBall,1),
1:size(footBall,2));

%Display the blurred image
figure, imshow(LPF_football, [])
% Display the Fourier Spectrum
% Move the origin of the transform to the center of the
frequency rectangle.
Fc=fftshift(F);
Fcf=fftshift(LPFS_football);
% use abs to compute the magnitude and use log to brighten
display
S1=log(1+abs(Fc));
S2=log(1+abs(Fcf));
figure, imshow(S1,[])
figure, imshow(S2,[])
```

Highpass filters:

sharpen (or shows the edges of) an image attenuate the low frequencies and leave the high frequencies of the Fourier transform relatively unchanged.

The highpass filter (**Hhp**) is often represented by its relationship to the lowpass filter (**Hlp**):

$$H_{hp}(u, v) = 1 - H_{lp}(u, v)$$

Because highpass filters can be created in relationship to lowpass filters, the following table shows the three corresponding highpass filters by their visual representations:

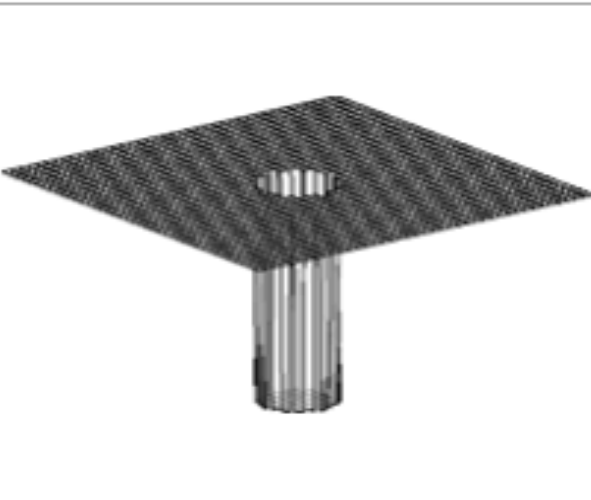
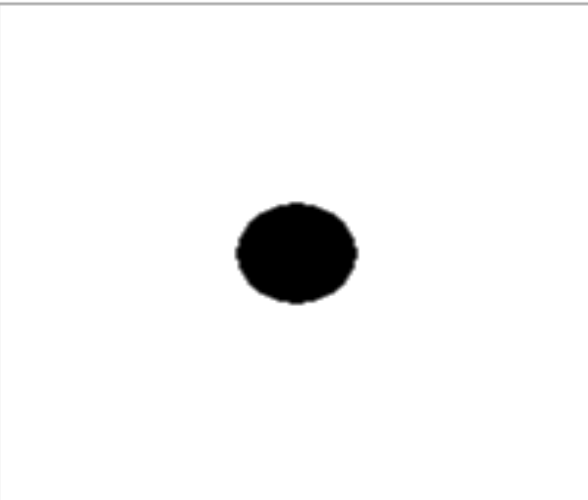
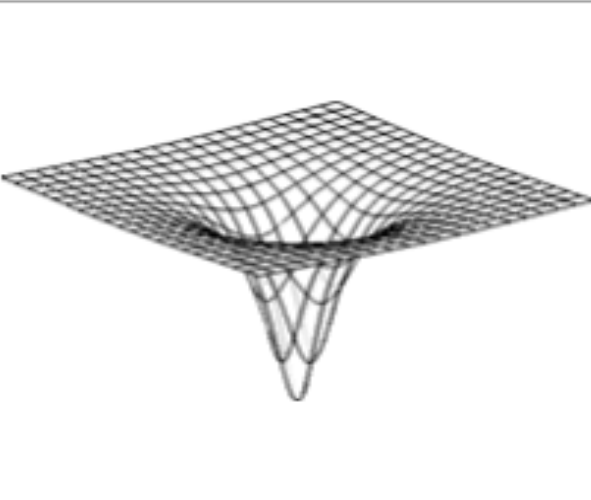
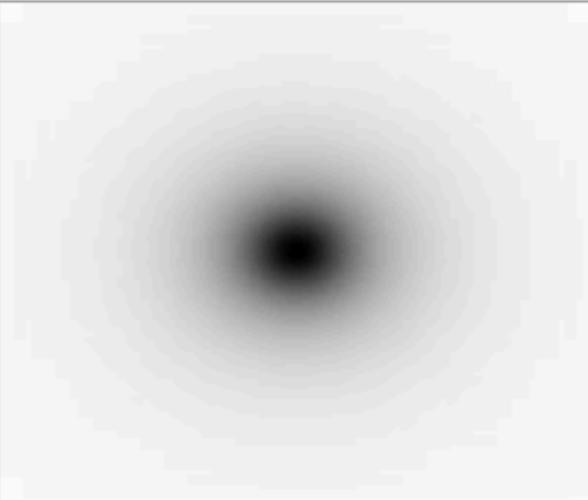
```

% The transfer function Hhp of a highpass filter is 1 - Hlp,
function H = hpfilter(type, M, N, D0, n)
%HPFILTER Computes frequency domain highpass filters
%   H = HPFILTER(TYPE, M, N, D0, n) creates the transfer function of
%   a highpass filter, H, of the specified TYPE and size (M-by-N).
%   Valid values for TYPE, D0, and n are:
%
%   'ideal'      Ideal highpass filter with cutoff frequency D0.  n
%                 need not be supplied.  D0 must be positive
%
%   'btw'        Butterworth highpass filter of order n, and cutoff D0.
%                 The default value for n is 1.0.  D0 must be positive.
%
%   'gaussian'   Gaussian highpass filter with cutoff (standard deviation)
%                 D0.  n need not be supplied.  D0 must be positive.
%
% where Hlp is the transfer function of the corresponding lowpass
% filter.  Thus, we can use function lpfilter to generate highpass
% filters.

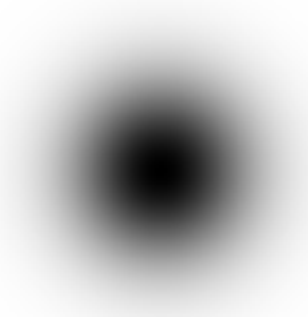
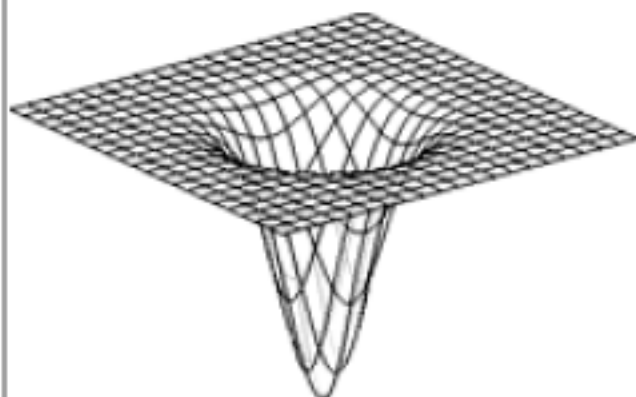
```

```
if nargin == 4
    n = 1; % Default value of n.
end

% Generate highpass filter.
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
```

Lowpass Filter	Mesh	Image
Ideal		
Butterworth		

Gaussian



Notch filters:

are used to remove repetitive "Spectral" noise from an image are like a narrow highpass filter, but they "notch" out frequencies other than the dc component attenuate a selected frequency (and some of its neighbors) and leave other frequencies of the Fourier transform relatively unchanged.

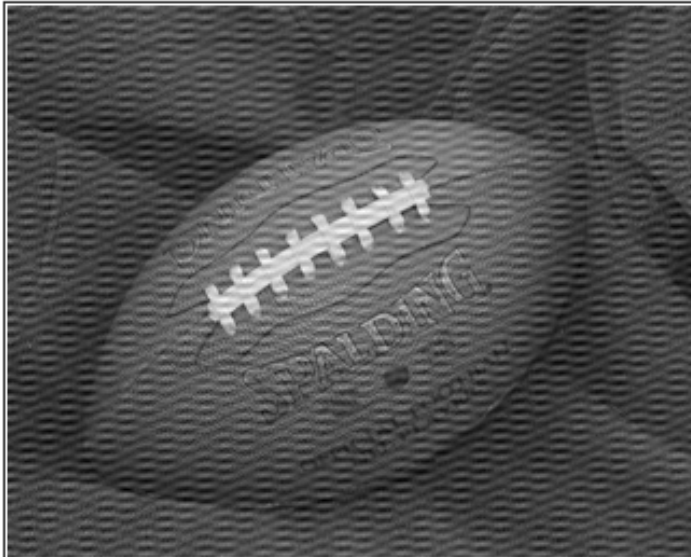
Repetitive noise in an image is sometimes seen as a bright peak somewhere other than the origin. You can suppress such noise effectively by carefully erasing the peaks. One way to do this is to use a notch filter to simply remove that frequency from the picture.

This technique is very common in sound signal processing where it is used to remove mechanical or electronic hum, such as the 60Hz hum from AC power.

Although it is possible to create notch filters for common noise patterns, in general notch filtering is an ad hoc procedure requiring a human expert to determine what frequencies need to be removed to clean up the signal.

The following is an example of removing synthetic spectral "noise" from an image.

Noisy Image



Fourier Spectrum of Image (noise peaks circled)

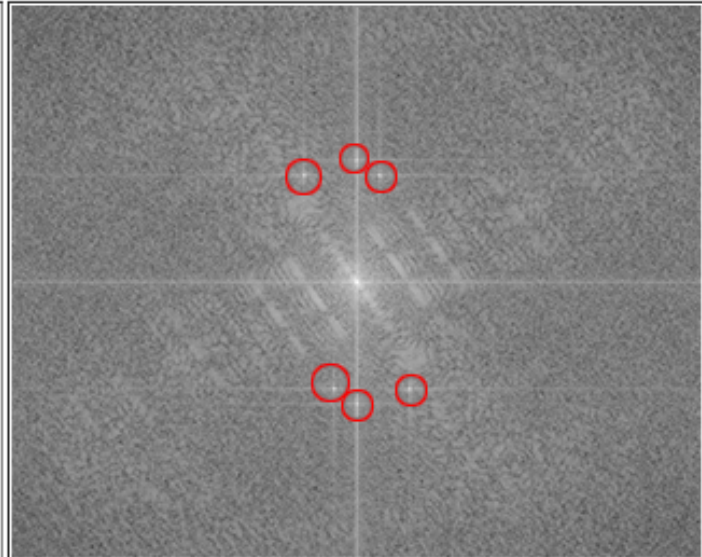
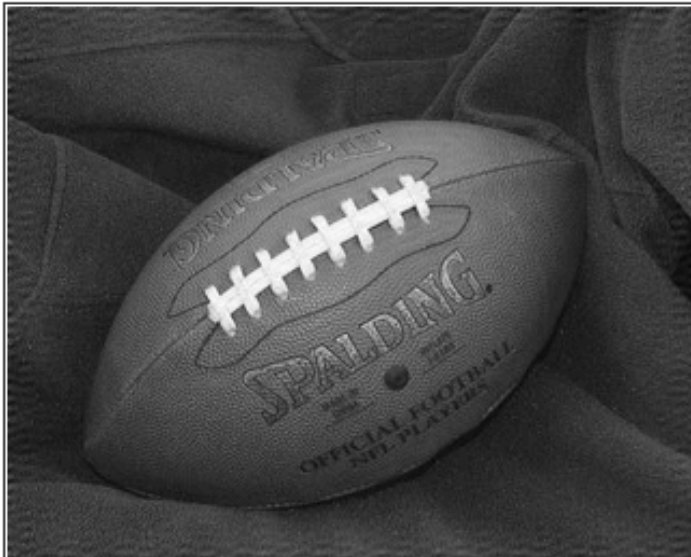
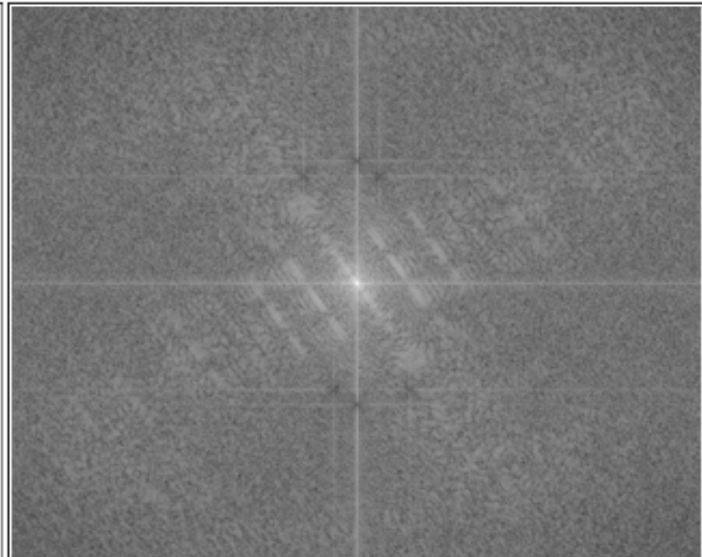


Image after Butterworth notch filters



Spectrum of image after Butterworth notch filters



```

function H = notch(type, M, N, D0, x, y, n)
%notch Computes frequency domain notch filters
%   H = NOTCH(TYPE, M, N, D0, x, y, n) creates the transfer function of
%   a notch filter, H, of the specified TYPE and size (M-by-N). centered at
%   Column X, Row Y in an unshifted Fourier spectrum.
%   Valid values for TYPE, D0, and n are:
%
%   'ideal'      Ideal highpass filter with cutoff frequency D0.  n
%                 need not be supplied.  D0 must be positive
%
%   'btw'        Butterworth highpass filter of order n, and cutoff D0.
%                 The default value for n is 1.0.  D0 must be positive.
%
%   'gaussian'   Gaussian highpass filter with cutoff (standard deviation)
%                 D0.  n need not be supplied.  D0 must be positive.
%
% The transfer function Hhp of a highpass filter is 1 - Hlp,
% where Hlp is the transfer function of the corresponding lowpass
% filter.  Thus, we can use function lpfilter to generate highpass
% filters.
if nargin == 6
    n = 1; % Default value of n.
end
% Generate highpass filter.
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
H = circshift(H, [y-1 x-1]);

```

```
footBall=imread('noiseball.png');
imshow(footBall)

%Determine good padding for Fourier transform
PQ = paddedsize(size(footBall));

%Create Notch filters corresponding to extra peaks in the Fourier transform
H1 = notch('btw', PQ(1), PQ(2), 10, 50, 100);
H2 = notch('btw', PQ(1), PQ(2), 10, 1, 400);
H3 = notch('btw', PQ(1), PQ(2), 10, 620, 100);
H4 = notch('btw', PQ(1), PQ(2), 10, 22, 414);
H5 = notch('btw', PQ(1), PQ(2), 10, 592, 414);
H6 = notch('btw', PQ(1), PQ(2), 10, 1, 114);
% Calculate the discrete Fourier transform of the image
F=fft2(double(footBall),PQ(1),PQ(2));
% Apply the notch filters to the Fourier spectrum of the image
FS_football = F.*H1.*H2.*H3.*H4.*H5.*H6;
```

```
% convert the result to the spacial domain.
F_football=real(ifft2(FS_football));

% Crop the image to undo padding
F_football=F_football(1:size(footBall,1), 1:size(footBall,2));

%Display the blurred image
figure, imshow(F_football,[])

% Display the Fourier Spectrum
% Move the origin of the transform to the center of the frequency
rectangle.
Fc=fftshift(F);
Fcf=fftshift(FS_football);

% use abs to compute the magnitude and use log to brighten display
S1=log(1+abs(Fc));
S2=log(1+abs(Fcf));
figure, imshow(S1,[])
figure, imshow(S2,[])
```

Exercises

Part 1: Identifying and Using High and Low Pass Filters

1. Download the following image "97.jpg" and store it in MATLAB's "Current Directory".



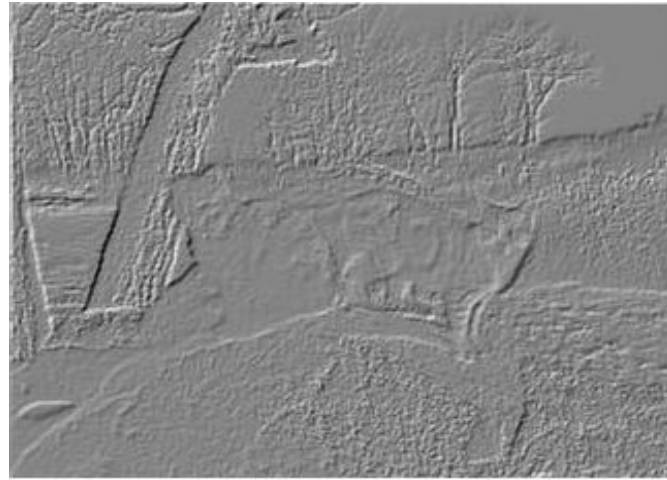
2. Identify which of the following is the result of a lowpass or highpass Butterworth filter and reproduce the results.



3. Display the Fourier spectrum for 97.jpg

Part 2: Using Spatial Filters in the Frequency Domain

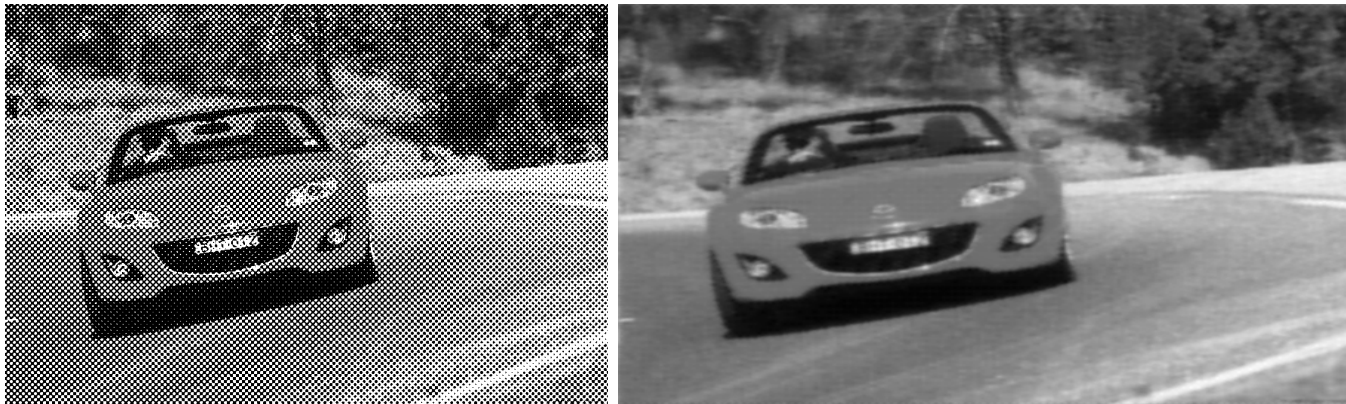
Download the following image "two_cats.jpg".



1. Load the image data.
2. Create a spatial filter to get the horizontal edge of the image
3. Create a spatial filter to get the vertical edge of the image (read the MATLAB documentation of fspecial).
4. Transform both of these filters to the frequency domain.
5. Transform the two_cats image to the frequency domain
6. Apply the appropriate operations in the frequency domain
7. Transform the data back into the spatial domain
8. Sum the horizontal and vertical edge components together
9. The resulting image should look like the image on the right.

Part 3: CSI Style Image Enhancement

You are the image processing expert for a local police department. A detective has reopened a cold case and part of his evidence is a newspaper print of a car. You have been asked to do some CSI style magic to see if you can learn the suspect's license plate number or see his face.



1. Use a set of notch filters to remove the peaks from the image's Fourier transform.

TIPS:

- create a function that takes a list of peaks as an argument
 - the peaks form a repetitive pattern. Figure out the pattern to save time.
2. Fine tune your results by trying varying widths of the three notch filter types. Provide a best effort solution for each one. Also provide a blurring based solution in Photoshop.

The image on the right side is one possible solution.