Lecture 4 Transfer function and image filtering

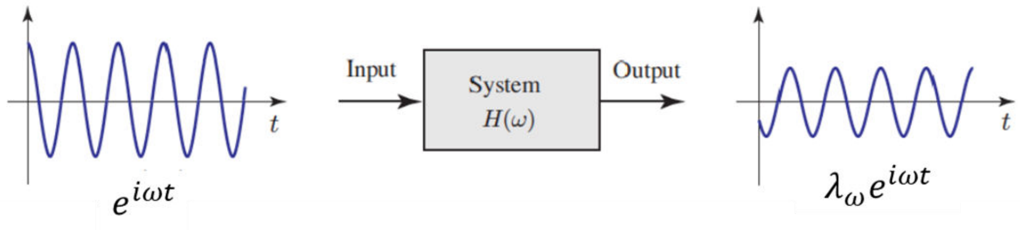
Table of Contents

In the previous lecture, we showed that the linear shift invariant (LSI) system's eigenfunctions are exponent functions. The signal can be conveniently decomposed on the eigenspace of the LSI system with Fourier transform, which introduces the concept of frequency analysis of the signal. We have also showed several properties of Fourier transform, particulary convolution-multiplication property. In this lecture, we will revisit the image filtering concepts and 'connect the dots'. The convolution kernel system transfer function in frequency domain, which are key concepts for system functions, such as optical transfer function (OTF) and modulation transfer function (MTF).

1. Transfer function of linear shift invariant system

1.1 Eigenfunction revisited

As discussed in the previous lecture, an eigenfunction of a system is a non-zero function that, when used as input signal, results in the output of itself scaled by a constant factor. For LSI system, the exponent function $e^{i\omega t}$is an eigenfunction, shown in ILL. 1.1.

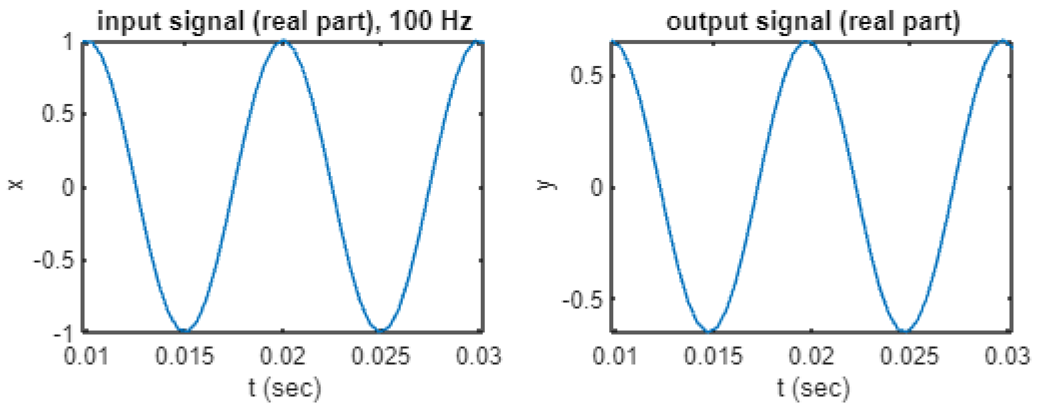


ILL. 1.1 *Illustration of the expoent function (the eigenfunction) input will result in exponent function output for LSI system.*

Example 1.1

Demonstration of LSI system gain by feeding in an exponent function.

```
nt = 1024; % number of time points
t_max = 0.5;% time span [Unit: sec]
t = linspace(0,t_max,nt);
dt = t(2)-t(1);
freq = 100; % frequency [Unit: Hz]
omega = 2*pi*freq; % angular frequency [Unit: rad/sec]
n_prd = ceil(1/freq/dt); % number of pixels per period.
x = exp(1i.*omega*t);% input
kernel = ones(1,10)./10; % 10 tap averaging filter.
y = conv(x,kernel,"same"); % output
figure;
subplot(121)
plot(t(n_prd:3*n_prd),real(x(n_prd:3*n_prd)));% show the 1st to 3rd period
xlabel('t (sec)'),ylabel('x');
title(['input signal (real part), ',num2str(freq),' Hz'])
subplot(122)
plot(t(n_prd:3*n_prd),real(y(n_prd:3*n_prd)));
xlabel('t (sec)'),ylabel('y');
title('output signal (real part)')
set(gcf,'Position',[100 100 600 200]);
```



When you slide the frequency of the input, you can see that the amplitude and the phase of the output changes. When the freqeuncy hits ~ 200 Hz, the amplitude is only about 2% of the input. But the frequency of the ouput is always the same as the input.

Note: It is not correct to say that harmonic functions (i.e.,sine and cosine) are eigenfunctions of LSI systems. If we have input $x(t) = \cos(\omega t)$, we expect a general form of output to be

$$x(t) \overset{\text{LSI}}{\longrightarrow} y(t) = A_\omega \cos(\omega t + \phi_\omega)$$

The output not only change the amplitude, but also has a phase shift. The output still has the same shape, but it does not has the form $\lambda_\omega x(t)$. Therefore the harmonic function is not an eigenfunction.

However, if you add an imaginary part with $\frac{\pi}{2}$ phase shift, then you construct the exponent function $x_e(t) = \exp(i\omega t)$. The shift is in the phase part of complex factor.

$$x_e(t) \xrightarrow{\text{LSI}} y_e(t) = A_\omega \exp(i\phi_\omega)\exp(i\omega t),$$

in which $\lambda_\omega = A_\omega \exp(i\phi_\omega)$ is the eigenvalue.

1.2 Transfer fucntion

As we can see in the precious section, this complex 'factor', λ_ω, can be treated as a function of frequency. We give it a formal name ***transfer function***: $H(\omega)$. You may have guessed why we use $H(\omega)$ here. Remember that in Lecture 2, we have concluded that any LSI system can be modeled as the input convolve with a kernel $h(t)$. The kernel uniquely determines the LSI system. The output

$$g(t) = h(t) \otimes f(t)$$

The input and output can be expressed in frequency domain:

$$g(t) = \int G(\omega)\phi_\omega(t)dt,$$

$$f(t) = \int F(\omega)\phi_\omega(t)dt,$$

where $F(\omega)$ and $G(\omega)$ are the Fourier transforms of $f(t)$ and $g(t)$, respectively. Since $\phi_\omega(t)$ is an eigenfunction of the system,

$$\lambda_\omega\phi_\omega(t) = h(t) \otimes \phi_\omega(t).$$

Therefore:

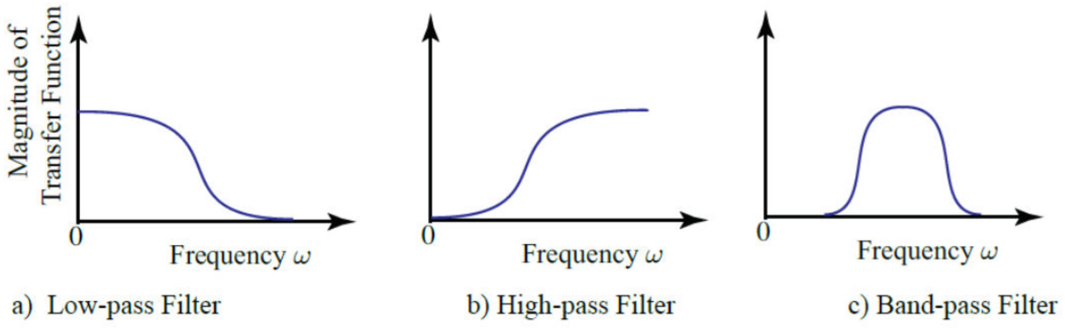
$$\int G(\omega)\phi_\omega(t)dt = h(t) \otimes \int F(\omega)\phi_\omega(t)dt$$
$$= \int F(\omega)[h(t) \otimes \phi_\omega(t)]dt = \int F(\omega)\lambda_\omega\phi_\omega(t)dt$$

$$\Rightarrow G(\omega) = \lambda_\omega F(\omega).$$

According to the Fourier convolution property: $G(\omega) = H(\omega)F(\omega)$, where $H(\omega)$ is the Fourier transform of the kernel, i.e. $h(t) \xrightarrow{\text{FT}} H(\omega)$. Then the eigenvalue of the system λ_ω is the Fourier transform of the kernel function. Kernel $h(t)$ is the inverse Fourier transform of $H(\omega)$, also called the point spread function (PSF) for 2D spatial coordinates.

It becomes clear that why the signal passing through a LSI system is equivalent to a 'filtering' process: Consider the input signal as superposition with all frequency components; the system enhance or supress each components determined by the transfer function $H(\omega)$. If $H(\omega)$ is 0, or some very small value at some ω, the output will not have these freqeuncy components, essentially being filtered out from the input. By performing Fouier transform of the kernel, we can characterize the spectral response of a LSI system.

Examples of transfer functions of three common types of filter are shown in **ILL. 1.2**.



ILL 1.2 *Transfer function of 1D low-pass, high-pass, and band-pass filters*

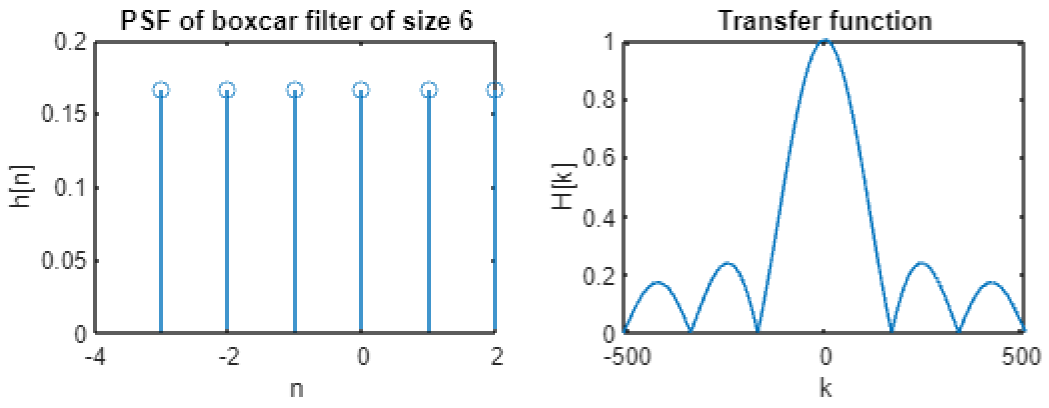
Example 1.2

1D Boxcar filter.

`B = padarray(A,padsize)` pads array `A` with an amount of padding in each dimension specified by `padsize`.

```
nt = 1024;
n_tap =6; % the size of boxcar filter
kernel = ones(1,n_tap)./n_tap;
pad_size = round((nt-length(kernel))/2);
kernel_pad = padarray(kernel,[0,pad_size],'both');
H = fftshift(fft(fftshift(kernel_pad)));
figure;
subplot(121);
```

```
stem((round(-n_tap/2):round(n_tap/2)-1),kernel);
title(['PSF of boxcar filter of size ', num2str(n_tap)]);
xlabel('n'), ylabel('h[n]');
subplot(122);
plot((round(-nt/2):round(nt/2)-1),abs(H));
title('Transfer function');
xlabel('k'),ylabel('H[k]');
set(gcf,'Position',[100 100 600 200]);
```

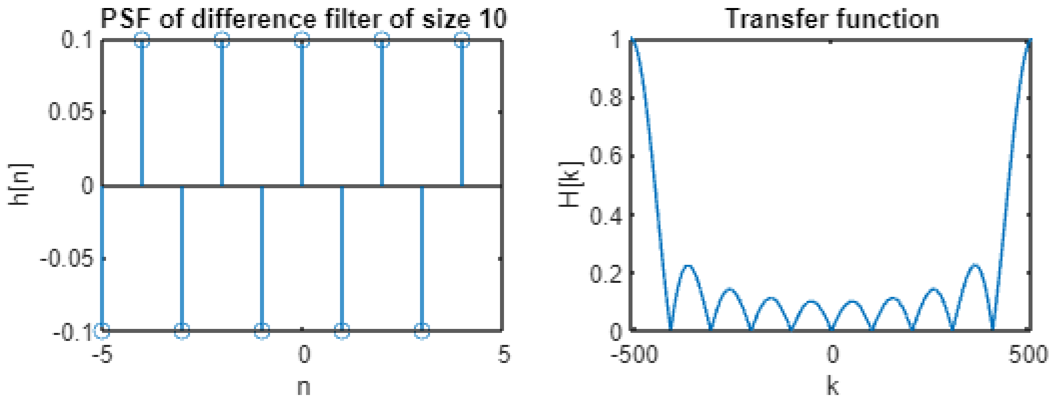


The transfer function shows a low-pass filter behavior with some side bands.

Example 1.3

1D difference filter.

```
nt = 1024;
n_tap =10; % the size of difference filter
kernel = repmat([-1,1],[1,n_tap/2])./n_tap;
pad_size = round((nt-length(kernel))/2);
kernel_pad = padarray(kernel,[0,pad_size],'both');
H = fftshift(fft(fftshift(kernel_pad)));
figure;
subplot(121);
stem((round(-n_tap/2):round(n_tap/2)-1),kernel);
title(['PSF of difference filter of size ', num2str(n_tap)]);
xlabel('n'), ylabel('h[n]');
subplot(122);
plot((round(-nt/2):round(nt/2)-1),abs(H));
title('Transfer function');
xlabel('k'),ylabel('H[k]');
set(gcf,'Position',[100 100 600 200]);
```

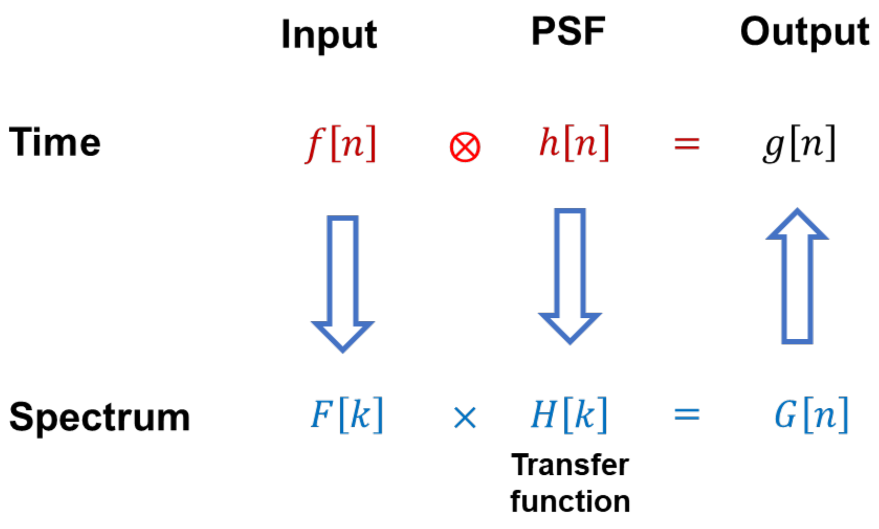


The transfer function shows a high-pass filter behavior with some side bands. Please adjust the length of the difference filter and observe the change of the passband.

1.3 Fast Fourier transform for convolution

According to Fourier convolution theorem, instead of directly calculate the convoution in time domain (red path in ILL. 1.3), we can carry the calculation in freqeuncy domain (blue path). It seems that the direct convolution is straightforward. However, if we consider the convolution of a PSF (kernel) of size N and the input of size N, the number of steps for relative shifting is $2N-1$. At step k, the number of multiplication is k. The total number of multiplication will be $\sum_{k=1}^{2N-1} k = 2N(N-1)$. The computational complexity is of order $O(N^2)$.

However, if we use spetral domain multiplication. The fast Fourier transform's computational complexity is $O(N\log_2 N)$. The multiplication in the spectral domain requires Nmultiplications. Though the calculation requires 3 FFT (blue arrow), the total computational complexity is still $O(N\log_2 N)$, which is lower than $O(N^2)$, particularly for large N.



ILL. 1.3 *Calculating convolution using Fourier convolution theorem.*

Example 1.4

Fast convolution using FFT

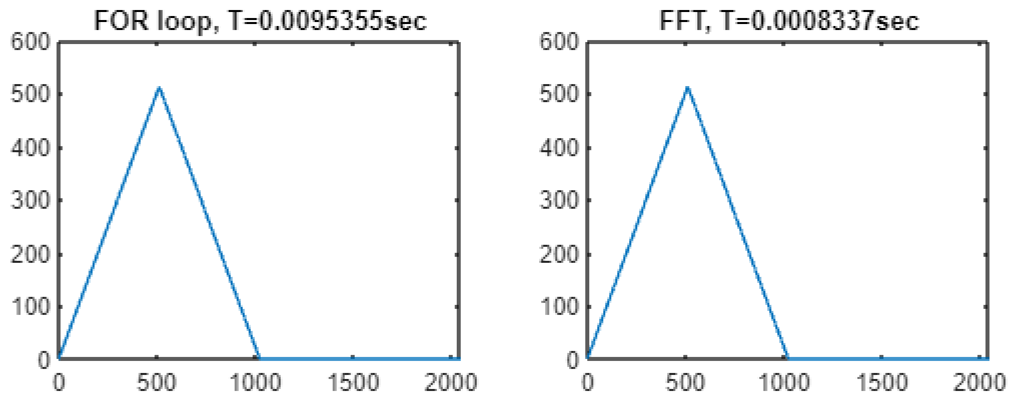
```
N = 1024;
f = zeros(1,N);
h = zeros(1,N);
w = 1:N/2; % width of rectangular function
f(w) = 1; % rectangular function as input
h(w) = 1; % rectangular function as kernel
g = zeros(1,2*N); % output

t0 = tic; % time the convolution
for ii = 1:N
    for jj = 1:N
        g(ii+jj) = g(ii+jj)+h(ii)*f(jj);
    end
end
del_t = toc(t0);

% matlab built-in convolution
g1=conv(h,f,'full');

% multiplication in frequency domain
h_f = padarray(h,[0,N],'post');
f_f = padarray(f,[0,N],'post');
t2 = tic;
```

```
g2 = ifft(fft(h_f).*fft(f_f));
del_t2 = toc(t2);
figure;
subplot(121)
plot(g)
title(['FOR loop, T=',num2str(del_t),'sec'])
subplot(122)
plot(abs(g2))
title(['FFT, T=',num2str(del_t2),'sec'])
set(gcf,'Position',[100 100 600 200]);
```

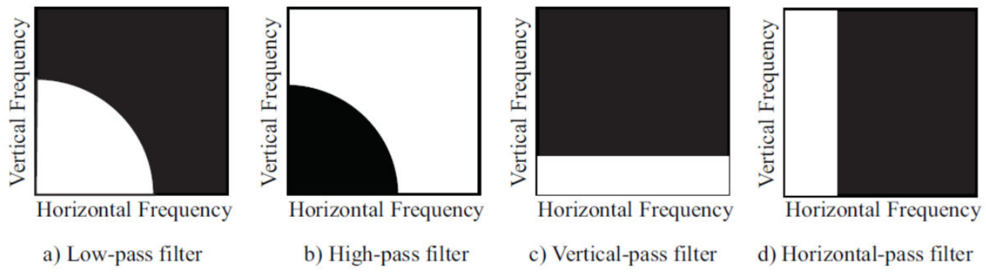


It is worth noting that using `for` loop is not efficient using interperative language such as MATLAB. However, it shows that convolution calculation in spectral domain is faster, especially when N is large and the two arrays are of comparible size. Please try the MATLAB built-in `conv` function, and check the time needed for excution.

2. Two-dimensional transfer function

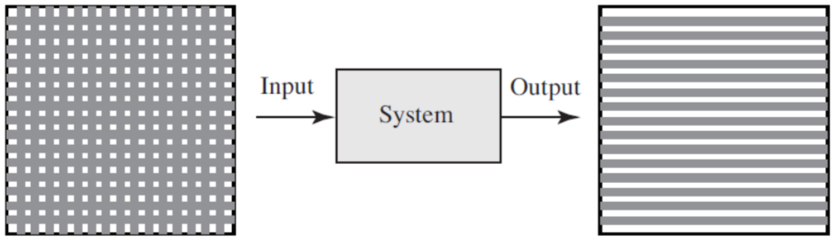
As in the 1D case, the convolution relation corresponding to multiplication in the Fourier domain, i.e., $G(k_x, k_y) = H(k_x, k_y)F(k_x, k_y)$, where k_x, k_y are the spatial frequency in x, y coordiates, respectively. $G(k_x, k_y)$, $H(k_x, k_y)$ and $F(k_x, k_y)$ are the 2D Fourier transforms of output $g(x, y)$, PSF $h(x, y)$, and input $f(x, y)$, respectively.

Similar to 1D transfer function we showed in the section before, low-pass, high-pass, and band-pass filter are the common types of filter. ILL. 2.1 shows some examples of the transfer functions of 2-D filter. ILL. 2.1 (a) and (b) shows isotropic filters whose transfer function is only a function of the radial coordinate, i.e. $H(\rho), \rho = \sqrt{k_x^2 + k_y^2}$. With the addtional dimension, the filtering can selectively filter the frequency components in a specific dimension, shown in ILL. 2.1 (c) and (d).



ILL. 2.1 *2-D image filters*

ILL. 2.2 is an illustration of selective filtering. The input is a superposition of a periodic function in vertical direction (y), and a periodic function in horizontal direction (x). The system is an horizontal low-pass filter. Transfer function shown in ILL. 2.1 (d) is an example of such. As a result, the output only contains the vertical periodic function.



ILL 2.1 *2-D filter illustration of horizontal low pass filtering*

In this section, we will take a look at the transfer functions we introduced in Lecture 2.

2.1 Transfer functions of additive 2D filters

We have mentioned some of the most popular additive 2-D filters. Specifically,

- Nearest-neighbor uniform averaging
- Boxcar filter
- Horizontal/vertical uniform averaging
- Non-uniform averaging

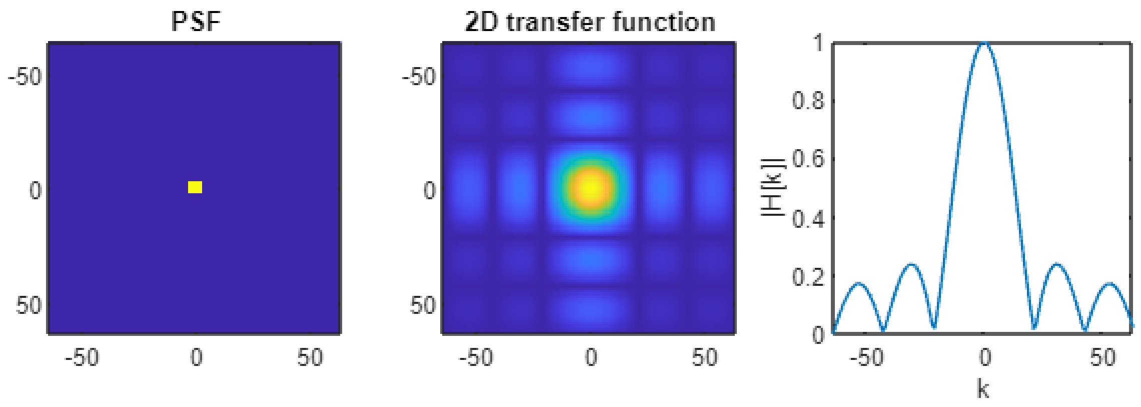
Example 2.1 and 2.2 are the examples of a boxcar filter and a 3x3 non-uniform averaging filter.

Example 2.1

2-D transfer function of Boxcar filter.

```
n = 128;
```

```
n_tap = 6;
h = ones(n_tap)./(n_tap.^2);
pad_size = round((n-length(h))/2);
h_pad = padarray(h,[pad_size,pad_size],'both'); % PSF
h_hat=fftshift(fft2(fftshift(h_pad))); % transfer function
figure;
subplot(131);
imagesc(-n/2:n/2-1,-n/2:n/2-1,h_pad);
axis image;axis equal;
title('PSF');
subplot(132);
imagesc(-n/2:n/2-1,-n/2:n/2-1,abs(h_hat));
axis image;axis equal;
title('2D transfer function');
subplot(133);
plot((-n/2:n/2-1),abs(h_hat(round(n/2),:)));
xlabel('k');ylabel('|H[k]|');
set(gcf,'Position',[100 100 700 200]);
```

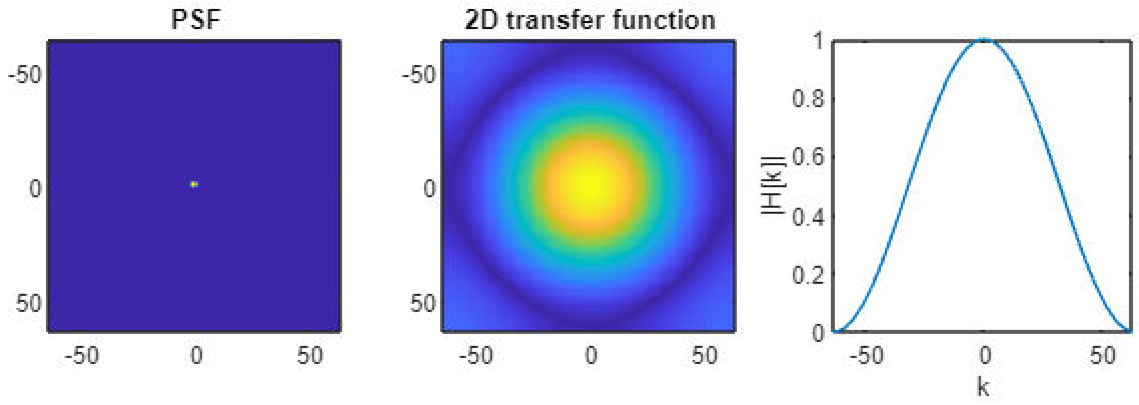


Example 2.2

2D transfer function of a non-uniform filter.

```
n = 128;
h = zeros(n);
h(n/2-1:n/2+1,n/2-1:n/2+1)=[0.05,0.15,0.05;0.15,0.2,0.15;0.05,0.15,0.05];
% PSF
h_hat=fftshift(fft2(fftshift(h))); % transfer function
figure;
subplot(131);
imagesc(-n/2:n/2-1,-n/2:n/2-1,h);
axis image;axis equal;
```

```
title('PSF');
subplot(132);
imagesc(-n/2:n/2-1,-n/2:n/2-1,abs(h_hat));
title('2D transfer function');
axis image;axis equal;
subplot(133);
plot(-n/2:n/2-1,abs(h_hat(round(n/2),:)));
xlabel('k');ylabel('|H[k]|');
set(gcf,'Position',[100 100 700 200]);
```



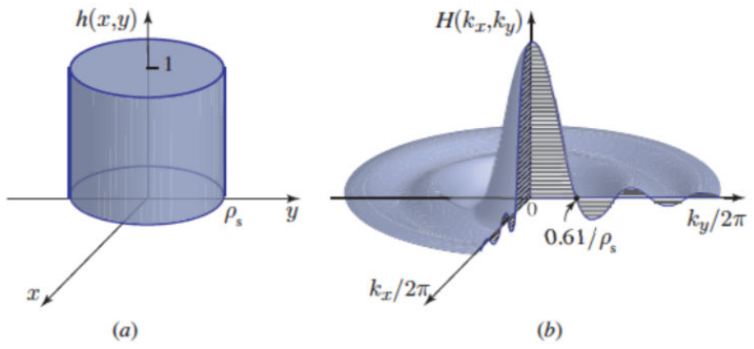
Example 2.3

Circ function PSF.

A system with an impulse response function in the form of a uniform circular function of radius ρ_s,

$$h(x,y) = \frac{1}{\pi(\rho_s)^2}\text{circ}\left(\frac{x}{\rho_s},\frac{y}{\rho_s}\right),$$

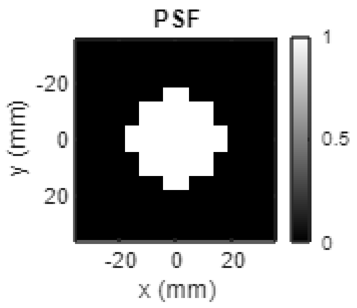
Transfer function: $H(k_x,k_y) = \dfrac{2J_1(\rho_s k_\rho)}{\rho_s k_s}, \quad k_\rho = \sqrt{k_x^2+k_y^2}$



ILL. 2.3 *(a) PSF is a circ function with radius ρ_s. (b) Transfer function has its first zero at a spatial frequency* $\frac{k_\rho}{2\pi} = \frac{0.61}{\rho_s}$ cycles/mm

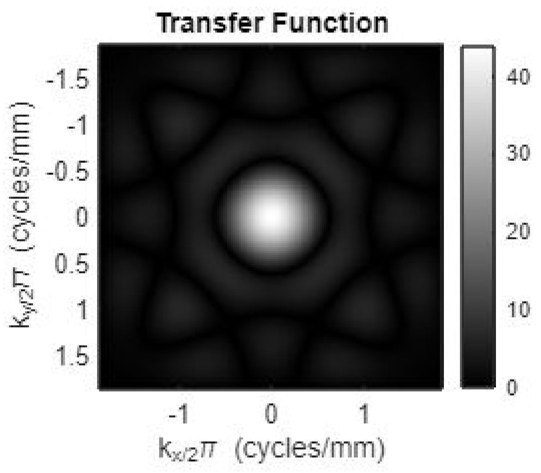
```
n_h = 8; % kernel half-size pixel number
xh = linspace(-2,2,2*n_h);
yh = linspace(-2,2,2*n_h);
dx = xh(2)-xh(1); % pixel size
[Xh,Yh]=ndgrid(xh,yh);
rho_s = 1; % PSF radius
Kernel=zeros(2*n_h);
Kernel(Xh.^2+Yh.^2<rho_s^2) = 1; % circle with radius of rho_s

figure;
imagesc(x,y, Kernel)
axis image;axis equal;
xlabel('x (mm)'),ylabel('y (mm)');
set(gcf,'Position',[100 100 200 200]);
colormap('gray');colorbar;
title('PSF')
```



```
% input image
InputImg=double(imread('cameraman.tif'));
[m,n]=size(InputImg); % size of the input image
x = (-m/2:m/2-1)*dx; % coordinates of the image
y = (-n/2:n/2-1)*dx;
% PSF and transfer function
kx = (-m/2:m/2-1)/m/dx;% spatial frequency coordinates
ky = (-n/2:n/2-1)/n/dx;
H = padarray(Kernel,[m/2,n/2],0); % PSF
H_hat = fftshift(fft2(fftshift(H))); % transfer function

figure;
imagesc(kx,ky,abs(H_hat));
axis image;axis equal;
xlabel('k_x/2\pi (cycles/mm)'),ylabel('k_y/2\pi (cycles/mm)');
set(gcf,'Position',[100 100 300 300]);
colormap('gray');colorbar;
title('Transfer Function')
```



You can see here the Transfer function has its first zero at a spatial frequency $\frac{k_\rho}{2\pi} = 0.61 \text{cycles}/_{\text{mm}}$

2.2 Denoising using low-pass filter

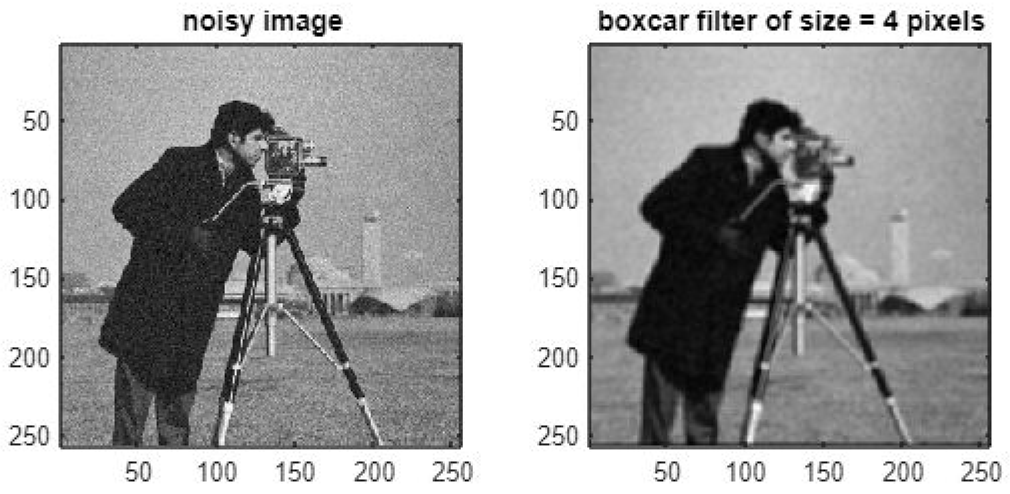
Noise often contains high-frequency components. This is especially true for electronic noise or environmental noise, which tends to add high-frequency 'hiss' or 'static' to the signal. By using a low-pass filter, high-frequency noise is attenuated, which means it's reduced in amplitude, making it less prominent in the output signal. This results in a cleaner or smoother signal. Low-pass filters are relatively easy to implement in both analog and digital forms, making them a convenient choice for many applications.

However, it is important to choose the right cutoff frequency for the low-pass filter. If the cutoff frequency is too low, it might also filter out important parts of the signal. If it's too high, it might not effectively remove the noise. This balance is crucial for effective noise reduction.

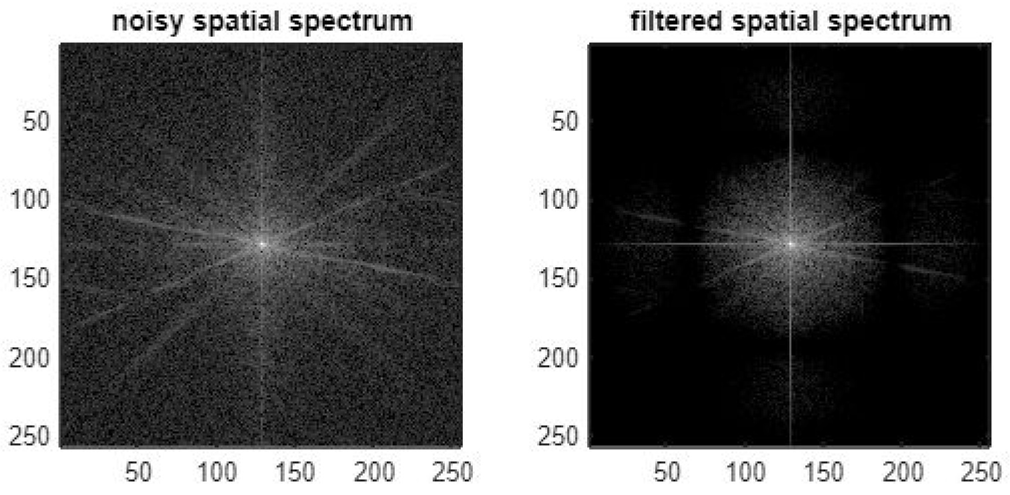
Example 2.4

Adjust the lowpass filter for denoising.

```
OriginalImg = double(imread('cameraman.tif'));
noise_level = 5e-2; % noise level
NoisyImg = OriginalImg+
noise_level*max(OriginalImg(:))*randn(size(OriginalImg));
% filter
n_tap =4;
h = ones(n_tap)./(n_tap.^2);
% filtered image
FltdImg=conv2(NoisyImg, h,'same');
figure;
subplot(121)
imagesc(NoisyImg);
axis image;axis equal;
title('noisy image')
colormap('gray');
subplot(122)
imagesc(FltdImg);
axis image;axis equal;
title(['boxcar filter of size = ',num2str(n_tap), ' pixels']);
colormap('gray');
set(gcf,'Position',[100 100 600 300]);
```



```
% spatial freqeuncy domain
figure;
subplot(121)
imagesc(abs(fftshift(fft2(fftshift(NoisyImg)))));
axis image;axis equal;
title('noisy spatial spectrum')
colormap('gray');
set(gca,'ColorScale','log') % plot log scale
subplot(122)
imagesc(abs(fftshift(fft2(fftshift(FltdImg)))));
axis image;axis equal;
title('filtered spatial spectrum')
colormap('gray');
set(gca,'ColorScale','log') % plot log scale
set(gcf,'Position',[100 100 600 300]);
```



Which filtered image looks the best? Can you come up with a metric to quantitatively compare their performances?

2.3 Subtractive 2D filters

Some of the most commonly used subtractive filters are:

- The horizontal difference operation
- The vertical difference operation
- Second order horizontal
- Second order two dimensional (Laplacian)

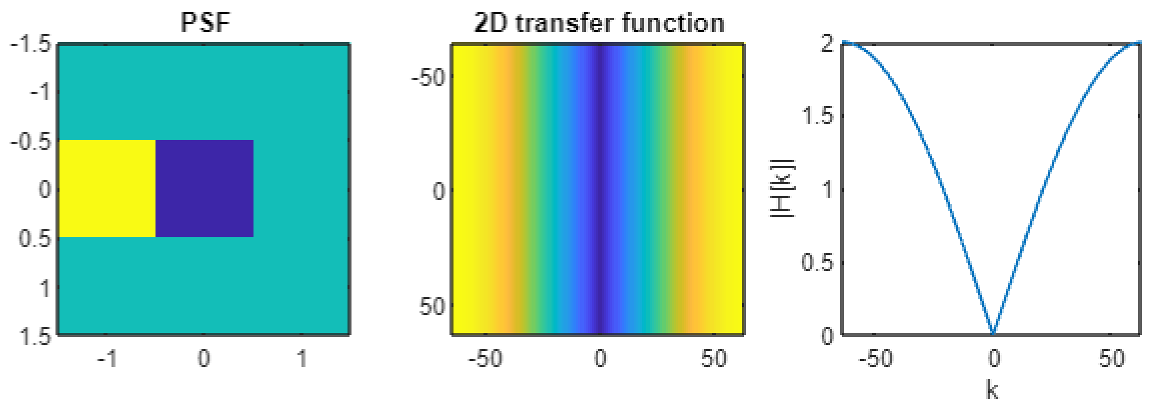
Example 2.5 and 2.6 are the examples of first order horizontal difference filter and a Laplacian filter.

Example 2.5

Transfer function of horizontal difference filter.

```
n = 128;
h_pad = zeros(n);
h_pad(n/2,n/2-1:n/2+1) = [1,-1,0];
h_hat = fftshift(fft2(fftshift(h_pad)));
figure;
subplot(131);
imagesc(-1:1,-1:1,h_pad(n/2-1:n/2+1,n/2-1:n/2+1));
```

```
axis image;axis equal;
title('PSF');
subplot(132);
imagesc(-n/2:n/2-1,-n/2:n/2-1,abs(h_hat));
axis image;axis equal;
title('2D transfer function');
subplot(133);
plot(-n/2:n/2-1,abs(h_hat(round(n/2),:)));
xlabel('k');ylabel('|H[k]|');
set(gcf,'Position',[100 100 700 200]);
```

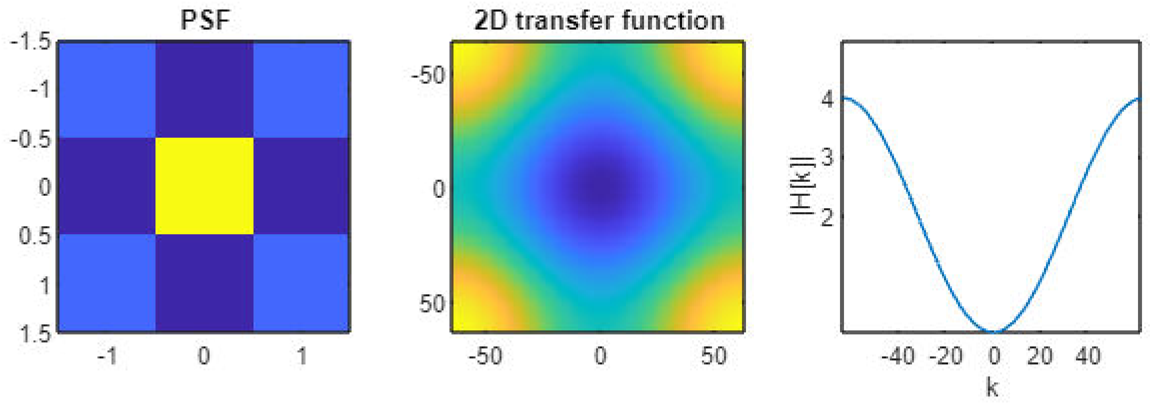


Example 2.5

Laplacian (2nd order difference) filter.

```
n = 128;
h_pad = zeros(n);
h_pad(n/2-1:n/2+1,n/2-1:n/2+1) = [0,-1,0;-1,4,-1;0,-1,0];
h_hat=fftshift(fft2(fftshift(h_pad)));
figure;
subplot(131);
imagesc(-1:1,-1:1,h_pad(n/2-1:n/2+1,n/2-1:n/2+1));
axis image;axis equal;
title('PSF');
subplot(132);
imagesc(-n/2:n/2-1,-n/2:n/2-1,abs(h_hat));
axis image;axis equal;
title('2D transfer function');
subplot(133);
plot(-n/2:n/2-1,abs(h_hat(round(n/2),:)));
xlabel('k');ylabel('|H[k]|');
```

```
set(gcf,'Position',[0 100 700 200]);
```



2.4 Edge detection using difference filtering

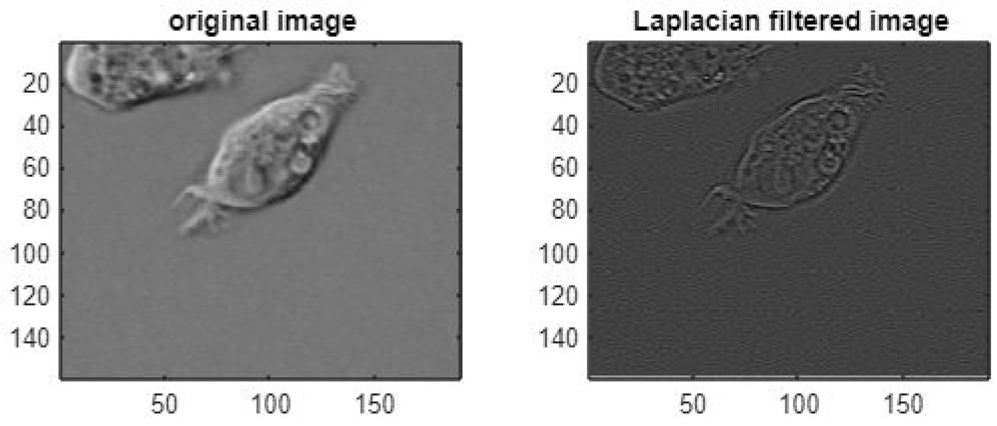
Edges in an image are a form of high-frequency component. Difference filters are sensitive to high-frequency changes, making them well-suited for detecting these rapid transitions. Difference filters can be designed to be sensitive to changes in specific directions (horizontal, vertical, diagonal), allowing for the detection of edges oriented in different ways. Difference filters also have some limitations. They can be sensitive to noise, as noise can also manifest as rapid intensity changes. To address this, it's common to apply a smoothing filter (like a Gaussian filter) before using a difference filter, helping to reduce the impact of noise on edge detection.

Example 2.7

Edge detection and enhancement using difference filtering.

```
InputImg = double(imread('cell.tif'));
h = [0,-1,0;-1,4,-1;0,-1,0];
HfImg=conv2(InputImg, h,'same');

figure;
subplot(121)
image(InputImg);
axis image;axis equal;
title('original image')
colormap('gray');
subplot(122)
imagesc(HfImg);
axis image;axis equal;
title('Laplacian filtered image');
colormap('gray');
set(gcf,'Position',[100 100 600 300]);
```



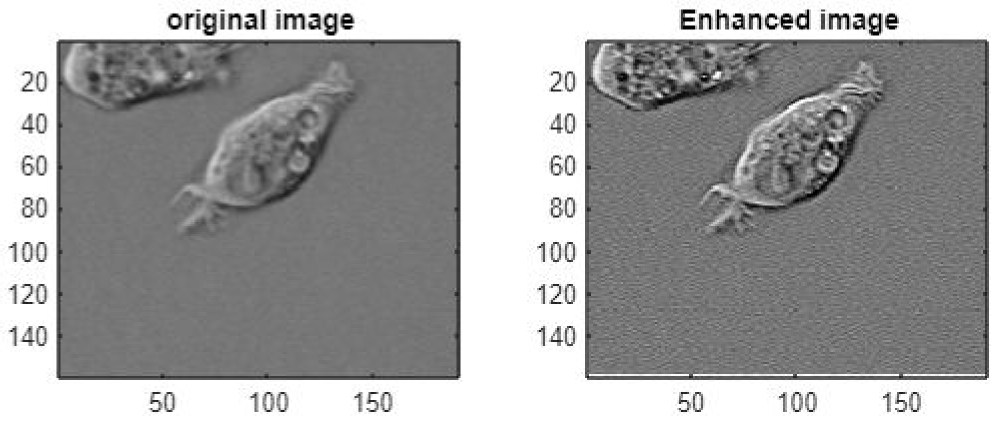
You may find the filtered image quite familiar. Actually, a special type of microscope called differential interference contrast (DIC) microscope can directly generate images similar to this one. The interested readers can search DIC modality and see how using interference can achieve high-pass filtering.

Now if we change the kernel a little bit, $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$, we get an edge enhancement filter.

```
InputImg = double(imread('cell.tif'));
h2 = [0,-1,0;-1,5,-1;0,-1,0];
EnhImg=conv2(InputImg, h2,'same');

figure;
subplot(121)
image(InputImg);
axis image;axis equal;
title('original image')
colormap('gray');
subplot(122)
image(EnhImg);
axis image;axis equal;
title('Enhanced image');
colormap('gray');
```

```
set(gcf,'Position',[100 100 600 300]);
```



Can you explain why?