

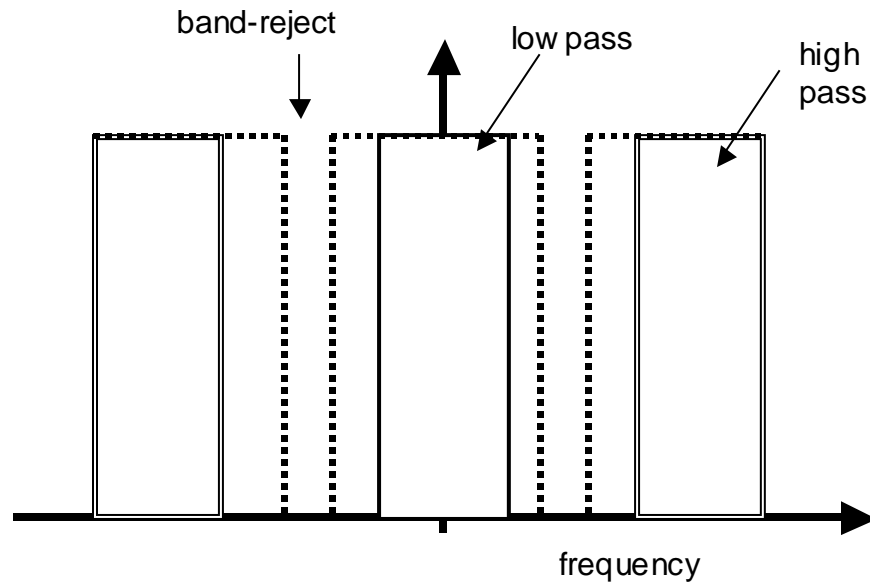
Lecture 8: Filters in Frequency Domain

If b is an image, then its Fourier transform will reflect the frequencies of the periodic parts of the image. By masking or filtering out the unwanted frequencies one can obtain a new image by applying the inverse Fourier transformation. A **filter** is a matrix with the same dimension as the Fourier transform of the padded image. The components of the filter usually vary from 0 to 1. If the component is 1, then the frequency is allowed to pass; if the component is 0, then the frequency is tossed out. Let $Filter$ represent such a matrix. Then the **filtered image** is given by $Newb$ in

$$Newb = \text{ifft2}(\text{Filter} .* \text{fft2}(b, 2 * nx - 1, 2 * ny - 1)).$$

The Matlab codes `fftsine.m` and `fftsine2d.m` in the previous lectures illustrated this for a low frequency sine wave with higher frequency sine “noise.”

Three important types of filters are low pass, high pass and band-reject. They are depicted in the following graphic where the low frequencies have been shifted to the center and one is viewing the diagonal section of the filter matrix.



The above filters have jumps and are often called “ideal” filters. The important filters that make use of polynomial and exponential approximations are the Butterworth and Gaussian filters. Let $w, d0$ be given and $\text{dist}(i,j) = ((i-(nx+1))^2 + (j-(ny+1))^2)^{1/2}$

Butterworth Band-reject Filter:

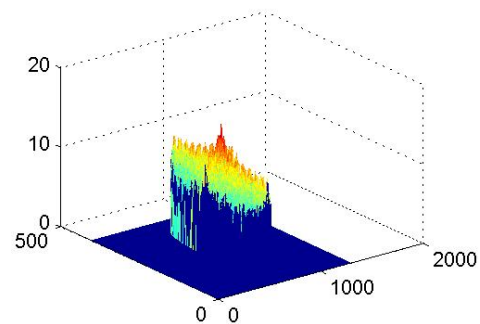
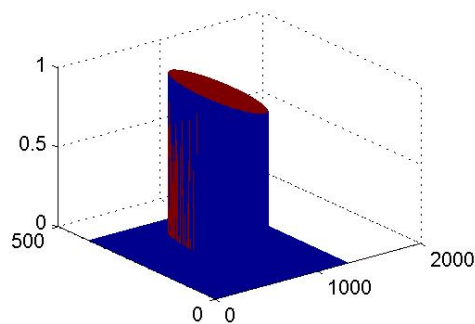
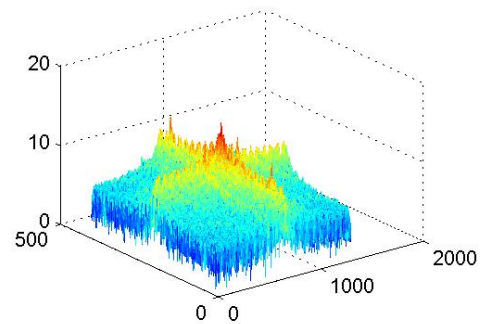
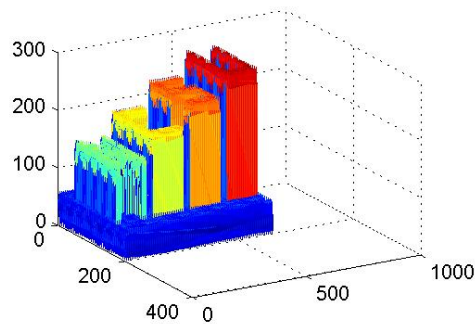
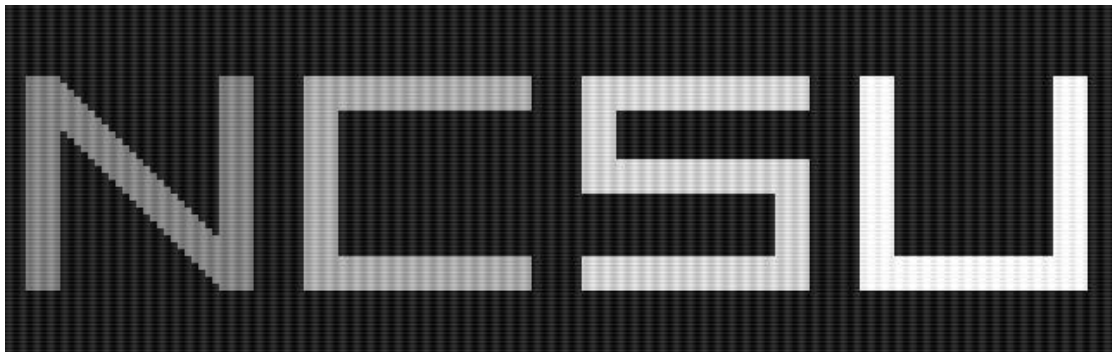
$$Filter(i, j) = \frac{1}{1 + (dist(i, j)w / (dist(i, j)^2 - d0^2))^{2n}}$$

Gaussian Band-reject Filter:

$$Filter(i, j) = 1 - e^{-\frac{1}{2}((dist(i, j)^2 - d0^2) / dist(i, j)w)^2}$$

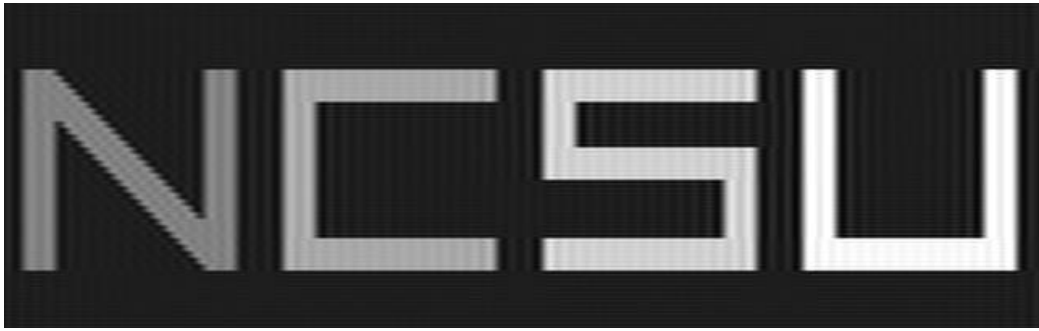
The parameters w and $d0$ control the width and location of the band. In the Butterworth band-reject filter the exponent n controls the steepness of the boundaries in the band of frequencies to be rejected. There are similar versions of high and low pass filters.

The Matlab code `filter_ncsu.m` uses a low pass filter to mask the noise from sine and cosine functions with frequency equal to 80 (160 in the padded Fourier transform).

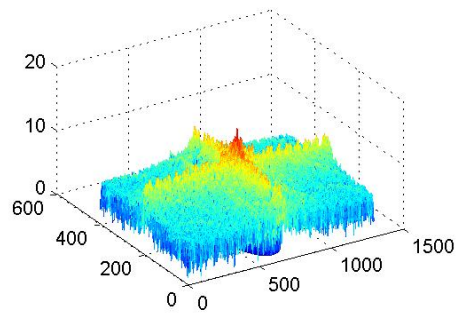
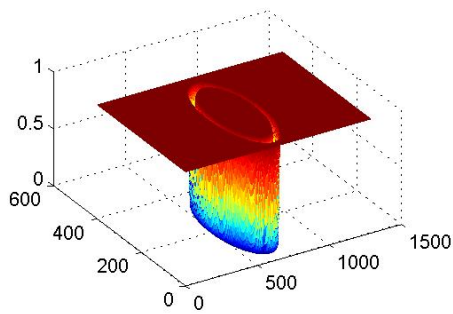
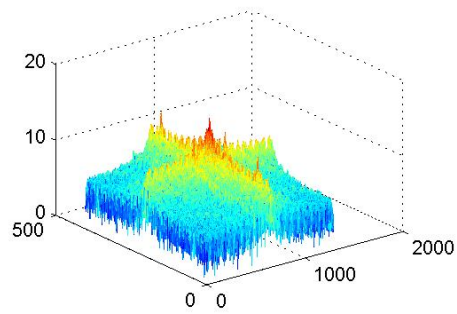
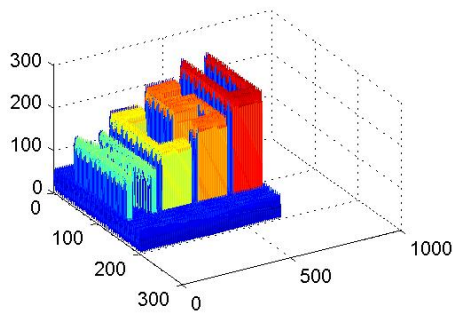


Matlab Code filter_ncsu.m

```
clear;
ncsu = [bign(5) bigc(7) bigs(9) bigu(11)];
newncsu = 20*ncsu;
[nx ny] = size(newncsu);
nx
ny
newncsu = newncsu(nx:-1:1,:);
newncsul = uint8(newncsu);
imwrite(newncsul, 'ncsu.jpg');
u = newncsu;
for i = 1:nx    % This is NCSU with periodic noise.
    for j = 1:ny
        u(i,j) = u(i,j) + ...
                15.*(1+sin(2*pi*((i-1)/nx)*80))+...
                15.*(1+sin(2*pi*((j-1)/ny)*80));
    end
end
sinencsu = uint8(u);
imwrite(sinencsu, 'sinencsu.jpg');
fftu = fft2(u,2*nx-1,2*ny-1);
fftu = fftshift(fftu);
subplot(2,2,1);
mesh(u');
subplot(2,2,2);
mesh(log(1+(abs(fftu))));
filter = ones(2*nx-1,2*ny-1);
d0 = 150;    % Use ideal low pass filter.
for i = 1:2*nx-1
    for j = 1:2*ny-1
        dist = ((i-(nx+1))^2 + (j-(ny+1))^2)^.5;
        if dist > d0
            filter(i,j) = 0;
        end
    end
end
subplot(2,2,3);
mesh(filter);
fil_ncsu = filter.*fftu;
subplot(2,2,4);
mesh(log(1+abs(fil_ncsu)));
fil_ncsu = ifftshift(fil_ncsu);
fil_ncsu = ifft2(fil_ncsu,2*nx-1,2*ny-1);
fil_ncsu = real(fil_ncsu(1:nx,1:ny));
fil_ncsu = uint8(fil_ncsu);
imwrite(fil_ncsu, 'sinencsu_fil.jpg');
```

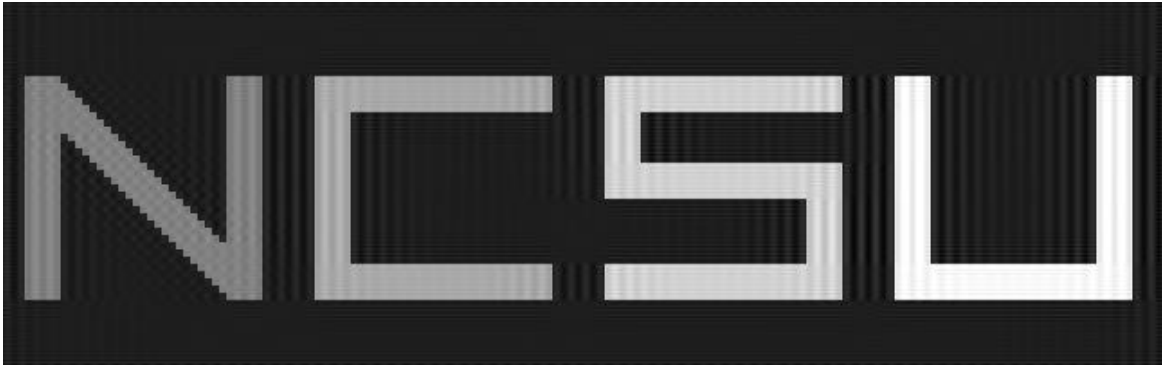


The filtered image `sinencsu_fil.jpg` is not entirely satisfactory. Another approach is to use a band-reject filter where one may need to experiment with the width, w , and the location, $d0$, of the band. The Matlab code `filter_bu.m` uses the Butterworth band-reject filter and may be applied to either the noisy big sine wave or the noisy `ncsu`.

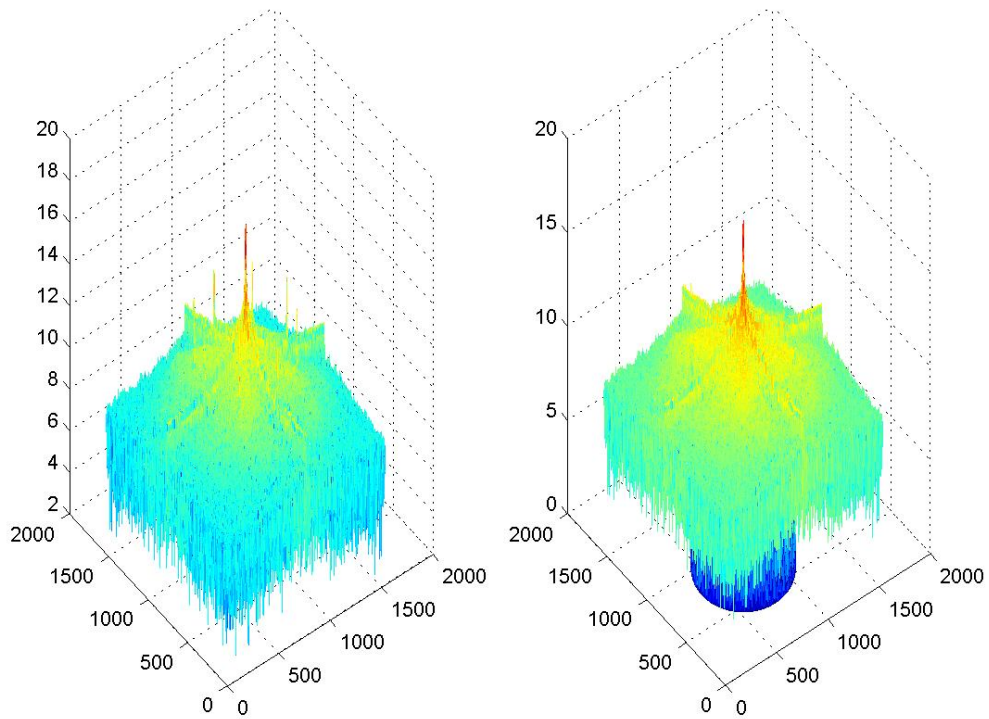


Matlab Code filter_bu.m

```
clear;
ncsu = [bign(5) bigc(7) bigs(9) bigu(11)];
newncsu = 20*ncsu;
[nx ny] = size(newncsu);
nx
ny
newncsu = newncsu(nx:-1:1,:);
newncsul = uint8(newncsu);
imwrite(newncsul, 'ncsu.jpg');
u = newncsu;
for i = 1:nx
    for j = 1:ny
        % This is the big wave with periodic noise.
        u(i,j) = 0+100*(1+sin(2*pi*((j-1)/ny)*5)) + ...
                15.*(1+sin(2*pi*((i-1)/nx)*80))+...
                15.*(1+sin(2*pi*((j-1)/ny)*80));
        % This is NCSU with periodic noise.
        %u(i,j) = u(i,j) + ...
                15.*(1+sin(2*pi*((i-1)/nx)*80))+...
                15.*(1+sin(2*pi*((j-1)/ny)*80));
    end
end
sinencsu = uint8(u);
imwrite(sinencsu, 'sinencsu.jpg');
fftu = fft2(u,2*nx-1,2*ny-1);
fftu = fftshift(fftu);
subplot(2,2,1);
mesh(u');
subplot(2,2,2);
mesh(log(1+(abs(fftu))));
filter = ones(2*nx-1,2*ny-1);
d0 = 160; % Use Butterworth band-reject filter.
n = 4;
w = 20;
for i = 1:2*nx-1
    for j = 1:2*ny-1
        dist = ((i-(nx+1))^2 + (j-(ny+1))^2)^.5;
        if dist ~= d0
            filter(i,j) = 1/(1 + (dist*w/(dist^2 - d0^2))^(2*n));
        else
            filter(i,j) = 0;
        end
    end
end
end
subplot(2,2,3);
mesh(filter);
fil_ncsu = filter.*fftu;
subplot(2,2,4);
mesh(log(1+abs(fil_ncsu)));
fil_ncsu = ifftshift(fil_ncsu);
fil_ncsu = ifft2(fil_ncsu,2*nx-1,2*ny-1);
fil_ncsu = real(fil_ncsu(1:nx,1:ny));
fil_ncsu = uint8(fil_ncsu);
imwrite(fil_ncsu, 'sinencsu_fil.jpg');
```



Another application of the band-reject filter is to the noisy aerial photograph. This image suffers from too much light an exposure and from banded sine and cosine noise. The light is modified by use of the power transformation with the power equal to two, and then the Butterworth band-reject filter is used to reduce to noise.



Matlab Code filter_aerial.m

```
clear;
aerial = imread('Fig3.09(a).jpg');
aerial = double(aerial);
[nx ny] = size(aerial);
nx
ny
u = aerial;

for i = 1:nx
    for j = 1:ny
        % This is aerial with periodic noise.
        u(i,j) = u(i,j) + ...
            5.*(1+sin(2*pi*((i-1)/nx)*200))+...
            5.*(1+sin(2*pi*((j-1)/ny)*200))+...
            5.*(1+cos(2*pi*((i-1)/nx+(j-1)/ny)*141))+...
            5.*(1+sin(2*pi*((i-1)/nx-(j-1)/ny)*141));;
    end
end
sineaerial = uint8(u);
imwrite(sineaerial, 'sineaerial.jpg');

c = 1.; % Use the power transformation to darken.
gamma = 2;
f_fp = 255*c*(u/255).^gamma;
u = f_fp;
fftu = fft2(u,2*nx-1,2*ny-1);
fftu = fftshift(fftu);
subplot(1,2,1)
mesh(log(1+(abs(fftu))));

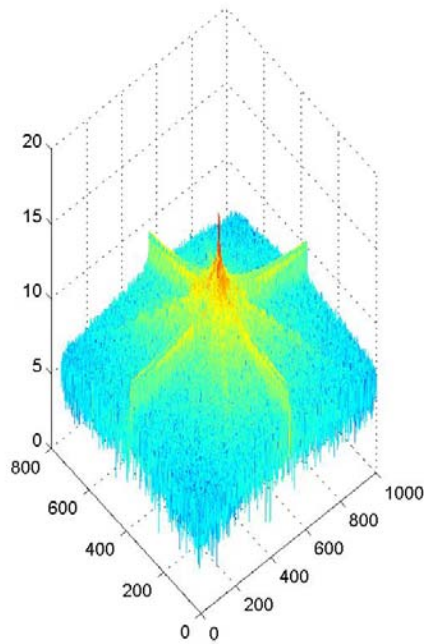
filter = ones(2*nx-1,2*ny-1);
d0 = 400; % Use Butterworth band reject filter.
n = 4;
w = 20;
for i = 1:2*nx-1
    for j = 1:2*ny-1
        dist = ((i-(nx+1))^2 + (j-(ny+1))^2)^.5;
        if dist ~= d0
            filter(i,j) = 1/(1 + (dist*w/(dist^2 - d0^2))^(2*n));
        else
            filter(i,j) = 0;
        end
    end
end
end
fil_aerial = filter.*fftu;
subplot(1,2,2)
mesh(log(1+abs(fil_aerial)));

fil_aerial = ifftshift(fil_aerial);
fil_aerial = ifft2(fil_aerial,2*nx-1,2*ny-1);
fil_aerial = real(fil_aerial(1:nx,1:ny));
fil_aerial = uint8(fil_aerial);
imwrite(fil_aerial, 'sineaerial_fil.jpg');
```

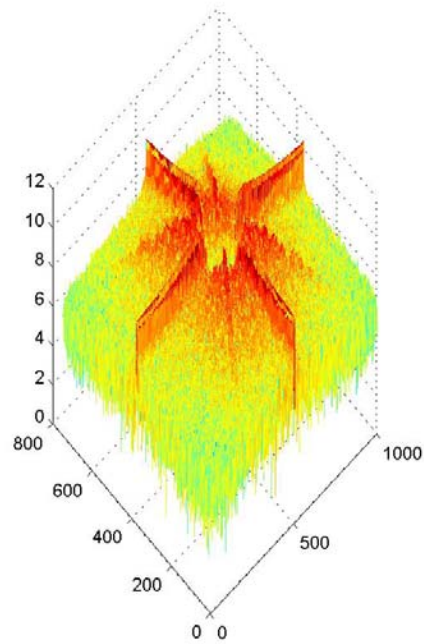


The Matlab code `filter_micro.m` uses a high pass filter to give emphasis to the higher frequencies in the image of a damaged electronic chip. The high pass image is then added to the original image so as to obtain a sharper image. The reader may find it interesting to experiment with width and frequency threshold of the Butterworth or the Gaussian high pass filters. Also, it is interesting to compare this sharpening, which is done in the frequency domain, with the sharpening done in the space domain as in the third lecture .

FFT of Image



High Pass Filter of FFT Image



Matlab Code filter_micro.m

```
clear;
micro = imread('Fig4.04(a).jpg');
micro = double(micro);
[nx ny] = size(micro);
nx
ny
u = micro;
micro = uint8(u);
imwrite(micro, 'micro.jpg');
fftu = fft2(u,2*nx-1,2*ny-1);
fftu = fftshift(fftu);
subplot(1,2,1)
mesh(log(1+(abs(fftu)))));
% Use Butterworth or Gaussian high pass filter.
filter = ones(2*nx-1,2*ny-1);
d0 = 100;
n = 4;
for i = 1:2*nx-1
    for j = 1:2*ny-1
        dist = ((i-(nx+1))^2 + (j-(ny+1))^2)^.5;
        % Use Butterworth high pass filter.
        filter(i,j) = 1/(1 + (dist/d0)^(2*n));
        filter(i,j) = 1.0 - filter(i,j);
        % Use Gaussian high pass filter.
        %filter(i,j) = exp(-dist^2/(2*d0^2));
        %filter(i,j) = 1.0 - filter(i,j);
    end
end
% Update image with high frequencies.
fil_micro = fftu + filter.*fftu;
subplot(1,2,2)
mesh(log(1+abs(fil_micro-fftu)));
fil_micro = ifftshift(fil_micro);
fil_micro = ifft2(fil_micro,2*nx-1,2*ny-1);
fil_micro = real(fil_micro(1:nx,1:ny));
fil_micro = uint8(fil_micro);
imwrite(fil_micro, 'micro_fil.jpg');
```

