Practice. Frequency domain filtering.

1. Theoretical Explanation

The Discret Fourier Transform (DFT) is infinitely periodic in *u* and *v* directions, determined by *M* and *N*: *f(x,y) = f(x+M,y) = f(x,y+N) = f(x+M,y+N)*. Despite this, DFT is calculated only in one period. The transform is symmetric respect the origin: *F(u) = F(-u)*. In 1-D case, fig. 1.a top, the period has a length *M* and it is centered in the origin. Because the 1-D DFT is implemented for only *M* points, it follows that computing the 1-D transform yields two *back-to-back* half periods in this interval. We are interested in obtaining one full, properly ordered period in the interval [0, *M*-1], moving the origin of the transform to the point *u = M*/2, fig. 1.a bottom. A similar situation exists with two-dimensional functions, and the origin must be moved to *u = M/2, v=N/2*, fig. 1.b.

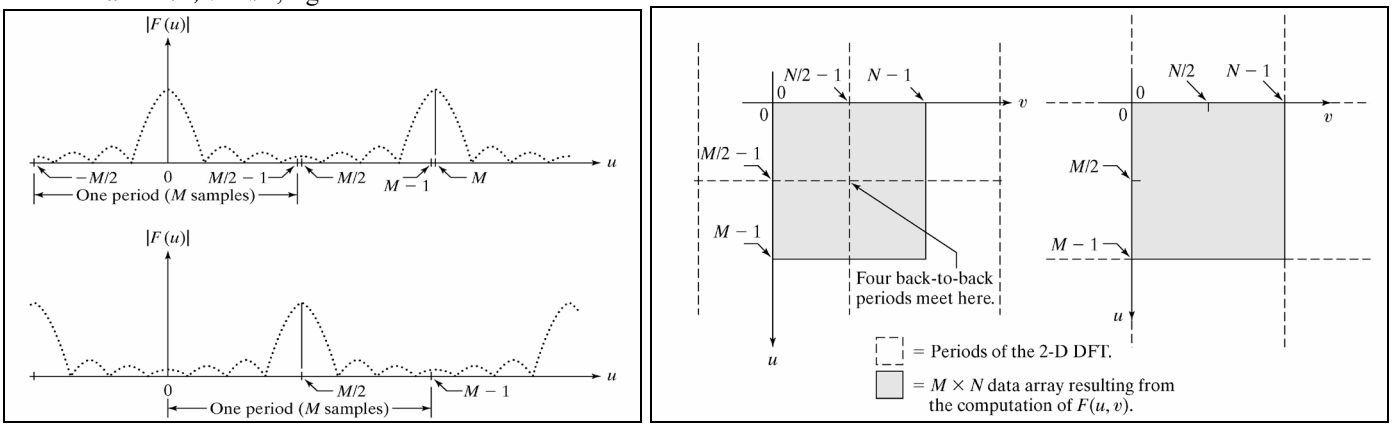

Figure 1. a) 1-D transform; b) 2-D transform.

Filtering in the Frequency Domain.

The foundation for linear filtering in both the spatial and frequency domain is the convolution theorem, which may be written as:

$$f(x,y) * h(x,y) \Leftrightarrow H(u,v)F(u,v) \quad \text{, and conversely,}$$
$$f(x,y)\,h(x,y) \Leftrightarrow H(u,v) * G(u,v)$$

The symbol '*' indicates convolution of the two functions, and the double arrow constitutes the Fourier transform. The first expression indicates that convolution of two spatial functions can be obtained by computing the inverse Fourier transform of the product of the Fourier transforms of the two functions.

In the spatial domain we convolve an image with a filter **mask** *h(x,y)*. We refer to *H(u,v)* as the **filter transfer** function. Fig. 2 shows the steps of image filtering within the frequency domain.

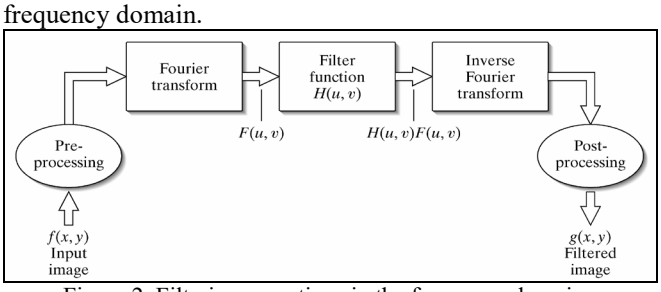

Figure 2. Filtering operations in the frequency domain.

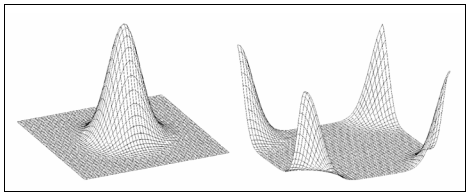

Figure 3. Lowpass filter transfer function, centered and non-centered.

Fig 3. shows the waveform of a lowpass filter transfer function.

To avoid convolving periodic functions that can cause interference between adjacent periods if the periods are close with respect to the duration of the nonzero parts of the funcions (*wraparound error)* we use the function PADDEDSIZE. In the header we can read the usage and result. If functions *f(x,y)* and *h(x,y)* are of size *AxB* and *CxD*, respectively, we can check that errors will be avoided if you choose to pad with zeros:

$$P \geq A + C - 1$$
$$Q \geq B + D - 1$$

Then, the syntax for the DFT after using paddedsize function is:

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

BASIC STEPS FOR FILTERING USING DFT

1. Obtain the padding parameters:
   `PQ = paddedsize(size(f));`
2. Obtain the Fourier transform.
   `F = fft2(f, PQ(1), PQ(2));`
3. Generate a filter function, H, of size PQ(1)xPQ(2). The filter must be in the format shown in fig.4 (right). If it is centered instead, as in fig.4 (left) let
   `H = fftshift(H)` before usign the filter.
4. Multiply the trasform 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));`

2. Practice Development

Computing and visualizing the 2-D DFT

The DFT and its inverse are obtained using a fast Fourier transform, FFT. The FFT of an *MxN* image array *f* is obtained as:

```
F= fft2(f);
```

This function returns a Fourier transform that is also of size *MxN*, with data arranged in the form shown in fig.1.b left. To avoid errors in filtering, it is necessary to pad the input image with zeros:

```
F= fft2(f, Q, P);
```

padding with so the resulting size is *PxQ* (using PADDEDSIZE-M function).
The Fourier spectrum is obtained by using:

```
S= abs(F);
```

which computes the magnitude (square root of the sum of the squares of the real and imaginary parts) of each element of the array.

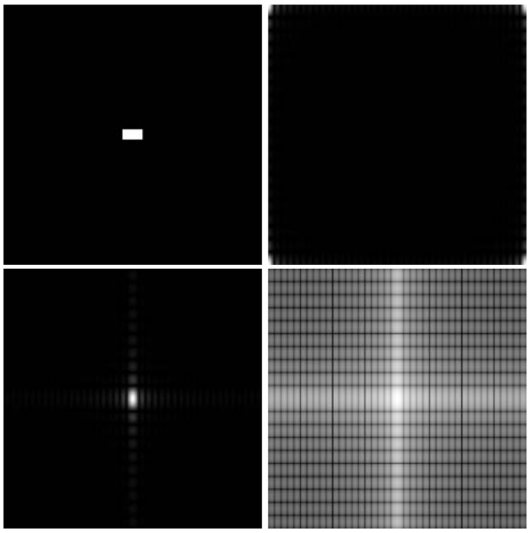

Figure 4. Spectrum visualization example.

Visual analysis of the spectrum by displaying it as an image is an important aspect of working in the frequency domain. Consider the simple image, *f*, in fig. 4. top left.We compute its Fourier transform and display the spectrum using:

```
>> F= fft2(f);
>> S = abs(F);
>> imshow(S, [ ])
```

Fig. 4 top right shows the result. The four bright spots in the corners of the image are due to the periodicity property mentioned above. To move the origin of the transform is:

```
>> Fc = fftshift(F);
>> imshow(Fc, [ ])
```

yielding the image in fig.4 bottom left; centering is evident.

Although the shift was accomplished as expected, the dynamic range of the values is so large (0 to 130500) compared to the 8 bits of the display that the bright values in the center dominate the result. The log transform is used:

```
>> S2 = log (1 + abs(Fc));
>> imshow(S2,[ ])
```

Fig. 4 bottom right is the result.
To invert the shift is used:

```
F= ifftshift(Fc);
```

The inverse Fourier transform is:

```
f = ifft2(F);
```

where F is the Fourier transform and f the resulting image. To avoid round-off errors is a good practice to extract the real part of the result after computing the inverse:

```
f= real(ifft2(F));
```

Exercise 4-1. Modify manualy the transform spectrum and show the results of the inverse transform. Is it any evident change in the original image?

Exercise 4-2. Check the effects in the inverse transform whether fftshift is executed over the spectrum or not.

Generating Filters Directly in the Frequency Domain

We focus on circularly symmetric filters that are specified as variuos functions of distance from the origin of the transform.

Creating Meshgrid Arrays for Use in Implementing Filters in the Frequency Domain

There is the need to compute distance functions from any point to a specified point in the frequency rectangle. Because FFT computations assume that the origin of the transform is at the top, left of frequency rectangle, our distance computations are with respect to that point. We give the function dftuv that provides the necessary meshgrid array for use in distance computation.

Exercise 4-3. Using the dftuv function:
1. find a meshgrid of 9x5 elements,
2. compute the distance squared from every point in this rectangle of size 9x5 to the origin of the rectangle,
3. obtain the same distances if the origin of the rectangle is moved to its center, which will be the coordinates of the center?

Lowpass Frequency Domain Filters H_{pb}

An *ideal lowpass filter* has the transfer function:

$$H(u,v) = \begin{cases} 1 \ if \ \ D(u,v) \le D_0 \\ 0 \ if \ \ D(u,v) > D_0 \end{cases}$$

where D_0 is a specified nonnegative number and *D(u,v)* is the distance from the point *(u,v)* to the center of the filter. The locus of points for which $D(u,v) = D_0$ is a circle. Keeping in mind that filter *H* multiplies the Fourier transform of an image, we see that an ideal filter cuts off (multiplies by 0) all components outside the circle and leaves unchanged all components on, or inside, the circle.

Other usual filters are:

A *Butterworth lowpass filter* of order n, with a cutoff frequency at a distance D_0 from the origin, has the transfer function:

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

The transfer function of a *Gaussian lowpass filter* is given by:

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

where σ is the standard deviation. By letting $\sigma=D_0$, we obtain the following expression in terms of the cutoff parameter D_0:

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

Exercise 4-4. Implement a function that allows to compute a lowpass filter with the following specifications:

```
function H = lpfilter(tipus, 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, DO 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.
%
%   'gauss'  gaussian lowpass filter with cutoff (standard
%              deviation) D0. n need not be supplied. D0 must
%              be positive.
```

Check that the result is correct with the sentences:

```
>> h=lpfilter('gauss',128,128,18);
>> surf(fftshift(h))
>> colormap(gray)
>> grid off
>> axis off
>> shading interp
```

and the result should be the same as in fig. 5.

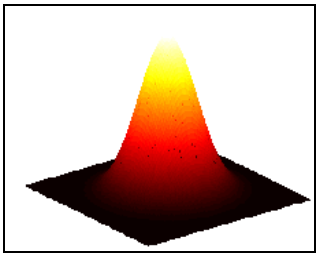

Figure 5. Gaussian lowpass filter.

Check the output generated with other filters, after and before the call fftshift. Below, in fig. 6 the 'ideal' and 'btw' order 3 filter, respectively.

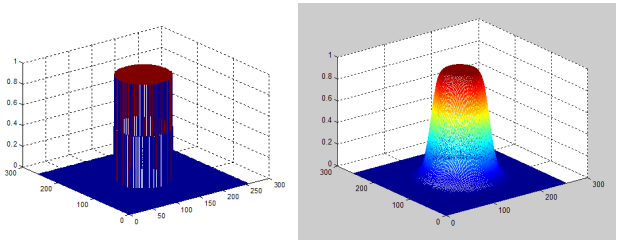

Figure 6. Lowpass filters examples.

Exercise 4-5. Filter with a lowpass filter the image 'moon.tif' according the following rules and show the results:

1. with an ideal filter
2. with a Butterworth filter order 2 and 4.
3. with a gaussian filter.

Sharpening Frequency Domain Filters

Highpass filters (H_{hp}) sharpens the image by attenuating the low frequencies and leaving the high frequencies of the Fourier tranform relatively unchanged. The transfer function of a highpass filter is:

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

Exercise 4-6. Implement a function (hpfilter) that allows to find the transfer function of a highpass filter, with exactly the same parameters that the function lpfilter.
Create some filters that look like the following:

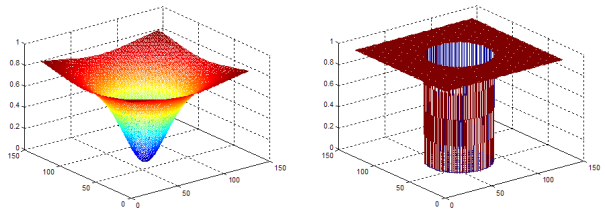

Figure 7. Highpass filter examples.

High-Frequency Emphasis Filtering

Highpass filters zero out the *dc* term, thus reducing the average value of an image to 0. An approach to compensate for this is to add an offset to a highpass filter, the high-frequency emphasis filter, with a transfer function:

$$H_{hfe}(u,v) = a + bH_{hp}(u,v)$$

where a is the offset, b is the multiplier and $H_{hp}(u,v)$ is the transfer function of a highpass filter.

Exercise 4-7. Filter the 'moon.tif' image with a high-frequency emphasis filter with a = 0.5 and b=2.0. Try to change the offset and the multiplier in order to improve the result.

Notch Filters to remove noise

Problem 1. A notch filter removes the unwanted frequency components at the designated point. When periodic noise at known frequencies is added to the images, the application of such filters can removed it. Suppose the image in Fig 8.

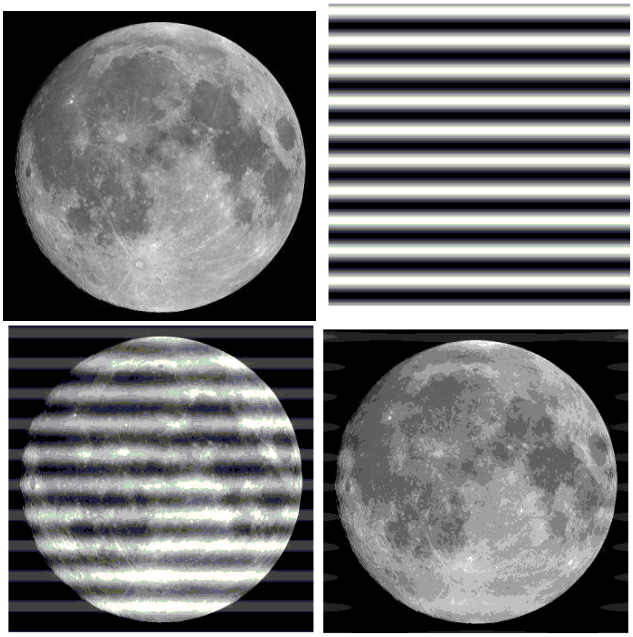

Fig 8. Notch fitlering in a noisy image.

In this problem we propose you to create a periodic noise (i.e. a sinus function in X along the Y axis) and add it to the original image. When FFT is applied, the noise appears as bright points in the spectrum, showing its position (frequency). The idea is to remove such points automatically knowing that if they are brighter than the others means that they have larger values respect the other spectrum coefficients. Figure 9 shows sthe spectrum of the noisy image before and after the peaks removal.

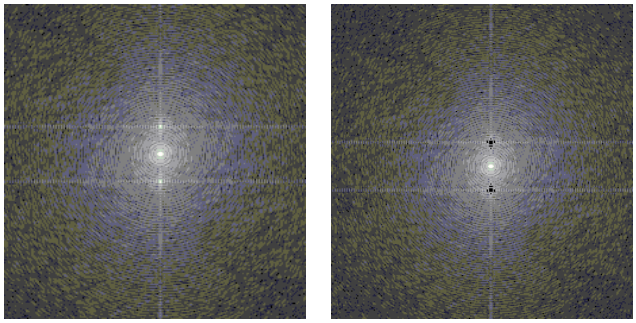

Fig. 9. Spectrum before and after noise peaks removal.

Procedure

1. Create a periodic noise.
2. Add noise to the original image.
3. Calculate FFT and visualize its power spectrum
4. Implement a function that automatically removes peaks corresponding to added noise
5. Apply inverse FFT to obtain a clean image.

Improvement: add noise at different frequencies and directions at the same time.

Where is the information? In the Magnitude or in the Phase?

Problem 2. In this problem we have to explore where the most of the information in a frequency transformation is, in the magnitude or in the phase of the spectrum?. To do so, you have to transform into the frequency domain two images, and then apply the inverse transformation with the magnitude of the first image and the phase of the second one, and viceversa, the magnitude of the second together with the phase of the first image. After analysing the results, please explain them after the inverse crossed transformation is done. Recall that a complex number x can be expressed as:

$$x = r.e^{j\theta}$$

where r and ϑ are the magnitude and phase respectively.

Tip: Use "truck.jpg" and "elephant.jpg" because they have the same size, and convert them to gray level values. The original, magnitude and reconstructed mixed images are shown in Fig. 10. Also, it is recommended to use the gray level version of both RGB images, but you can check the results in color too.

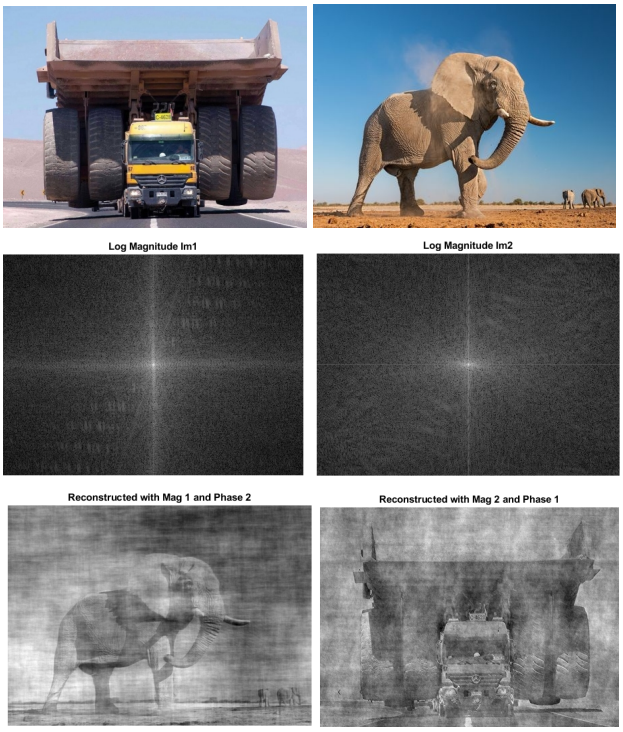

Fig. 10. In every row: Original images, magnitude spectra and crossed reconstruction.

SUPPLIED FILES: PADDEDSIZE.M; DFTFILT.M; DFTUV.M