①

Computing and Visualizing the 2-D DFT in Matlab

The DFT and its inverse are obtained using a fast fourier transform (FFT) algorithm, The FFT of an M×N image f is obtained using the fft2 function, with two syntaxes:

1) F = fft2(f) :

This function returns a fourier transform, size M×N, with data arranged with the origin of the data at the top left, and with four quarter periods meeting at the center of the frequency rectangle.

2) F= fft2 (f, P, Q) .

fft2 pads the input with the required number of zeros, so the resulting transform is of size P×Q (this padding is necessary when the Fourier transform is used for filtering).

The Fourier spectrum is obtained by using function abs :

S = abs(F)

which computes the magnitude of each element of the array. Visual analysis of the spectrum by displaying it as an image is an important aspect of working in the frequency domain.

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

IPT function fftshift can be used to move the origin of the transform to the center of the frequency rectangle; the syntax is:

```
>> Fc = fftshift( F) .
```

* The dynamic range of the values in this spectrum is so large (0 to 20400), this difficulty is handled via a Log transformation. :

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

Function ifftshif reverses the centering, its syntax is

F = ifftshift(Fc).

This function can be used to convert a function that is initially centered on a rectangle to a function whose Center is at the top left corner of the rectangle.

Centering the frequency rectangle :

1- the center of the frequecny rectangle is at $(\frac{M}{2}, \frac{N}{2})$ if the variables u and v run from 0 to M-1 and N-1 respectively.

• the center of an 8×8 frequency square is at point (4,4), which is the fifth point along each axis, counting up from (0,0)

• In Matlab : the variables run from 1 to M and 1 to N the center is $[\frac{M}{2}+1, \frac{N}{2}+1]$: in 8×8 example, the center (5,5)

2- if M and/or N are odd :

the Center for Matlab computations is obtained by rounding M/2 and N/2 down to the closest integer.

example : 7×7 region → the Center will be (3,3) if we Count from (0,0) and (4,4) if we count up from (1,1).

Using Matlab's function floor, the center of the frequency rectangle for Matlab computations is at :

[floor(M/2) + 1, floor(N/2) + 1]

* inverse Fourier transform is computed using function ifft, which has the following syntax :

a) » f = ifft2(F).

F - Fourier transform , f - resulting image.

Note : if the input used to compute F is real, the inverse in theory should be real, In practice, the output of ifft2 has small imaginary components resulting from round-off errors.

To extract real part, after computing the inverse, we use :

» f = real(ifft2(F));

b) f = ifft2(F, P, Q) - pads F with zeros, so that the size is P × Q before computing the inverse.

Filtering in the frequency domain:

Filtering in the spatial domain consists of convolving an image $f(x,y)$ with a filter mask $h(x,y)$.
According to the convolution theorem, we can obtain the same result in the frequency domain by multiplying $F(u,v)$ by $H(u,v)$ (the fourier transform of the spatial filter is called filter transfer function).
the idea in frequency domain filtering is to select a filter transfer function that modifies $F(u,v)$ in a speified manner.

- Low pass filter :
  - ideal Lowpass filter (ILPF)
  - Butterworth Lowpass filter (BLPF)
  - Gaussian Lowpass filter (GLPF)
- High pass frequency domain filters:
  - IHPF (ideal High pass)
  - BHPF (Butterworth high pass)
  - GHPF (Gaussian).

- padding issue:

images and theire transforms are automatically considered periodic, if we select to work with DFTs to implement filtering, which cause interference between adjacent periods, this interference called wraparound error.
To avoid wraparound error it is necessary to extend (pad) the input functions. In the case of two functions $f(x,y)$ and $h(x,y)$ of sizes $A \times B$ and $C \times D$ respectively, we should append zeros to the rows and columns of both so that they are of size $P \times Q$ where

$$P \geq A + C - 1 \quad \text{and} \quad Q \geq B + D - 1.$$

if the functions of the same size $M \times N$ then we use the following padding values:

$$P \geq 2M - 1 \quad \text{and}$$
$$Q \geq 2N - 1$$

Example 1 : write a function, called paddedsize that computes the parameters P and Q of the size for padding (P x Q) needed for input image, you may add optional parameters to this function like :

- Compute the minimum even values of P and Q satisfying the equations for padding.
- pad the image to form square images of size equal to the nearest integer power of 2 (efficient implementation of the FFT can be used).

```
function PQ = Padded size (AB, CD, PARAM)
% PADDEDSize computes padded size useful for FFT-based filtering
%   PQ = PADDEDSIZE (AB), where AB is a two-element size
%   vector, this option 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.
```

* P = nextpow2(n) - returns the smallest integer power of 2 that is greater than or equal the absolute value of n.
* with PQ thus computed using function paddedsize, we use the following syntax for fft2 to compute the FFT using zero padding :

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

* filtering without padding :

```
f = imread('square.tif');
imshow(f, []);
[M N] = size(f);
F = fft2(f);
sig = 10;
H = lpfilter('gaussian', M, N, sig);
G = H.*F;
g = real(ifft2(G));
figure, imshow(g, []).
```

% filtering with padding

```
PQ = paddedsize(size(f));
Fp = fft2(f, PQ(1), PQ(2));
% Compute the FFT with padding
Hp = lpfilter('gaussian', PQ(1), ...
         PQ(2), 2*sig);
Gp = Hp.*Fp;
gp = real(ifft(Gp));
gpc = gp(1:size(f,1), 1:size(f,2));
imshow(gpc, []).
```

Cropping the image to the original size.

Note : We use 2* sig here because the filter size is now twice the size of filter without padding.

A similar result ~~should~~ would be obtained by performing the following spatial filtering :

```
h = fspecial (' gaussian', 15, 7);
gs = imfilter (f, h); % pads the border of the image with 0s.
```

* Basic Steps in DFT Filtering.

The basic steps in DFT filtering in MATLAB are :

1) obtain the padding parameters using function paddedsize

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

2) obtain the fourier transform with padding

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

3) Generate a filter function, H, of size PQ(1) x PQ(2). if the filter is centered then let H= fftshift (H), befor using it

4) Multiply the transform by the filter :

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

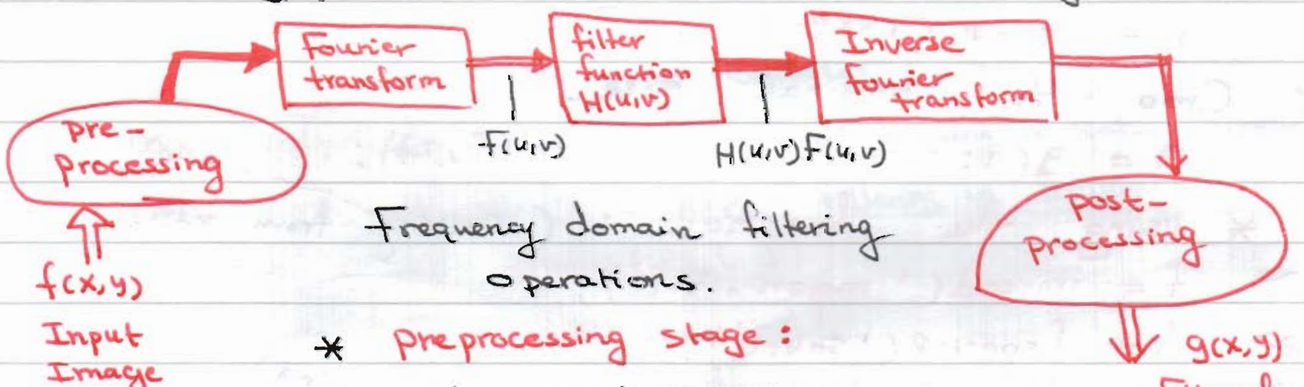
5) obtain the real part of the inverse FFT of G :

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

6) Crop to the original size :

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

This filtering procedure is summarized in the following figure :



Frequency domain filtering operations.

* preprocessing stage :
  - determine image size.
  - obtaining the padding parameters.
  - generating a filter.

* postprocessing stage :
  - Computing the real part.
  - Cropping the image.
  - Convert image to Uint8 or Uint16.