18-Aug-2023 U3A Maths Term 3 / Week 5

D.F.T. Application – Image Processing

Last week we looked at the application of Discrete Fourier Transform (DFT) for a set of values in a one dimensional array. This can be used for audio signal analysis, audio filtering, data compression, etc. For illustration of DFT calculations, the Octave script included the actual equations for DFT and inverse DFT.

Octave provides a much simpler and more efficient command for performing DFT calculations. It is called the Fast Fourier Transform or FFT in short. An example is shown below:

```
>> f = fft(x);            ( For x = [1,1,1,1]  The result is f = [4,0,0,0] )
```

Inside the computer, a black & white image is stored as a two dimensional matrix. Each item in the matrix represents the colour intensity of a pixel ('dot') on the screen. A value of 0 corresponds to 'black' and a value of 255 corresponds to 'white'. The values in between correspond to various shades on 'grayscale'.

A colour image needs to store 3 colours (Red, Green, Blue) for each pixel. Hence, the matrix will be a 3 dimensional matrix. For simplicity, we will use a black & white image to illustrate image compression using DFT. Octave provides a function to convert a standard colour image in JPEG format into a black & white image.

Octave also provides a function to perform Fast Fourier Transforms for a matrix, which is useful for image processing. An example is shown below:

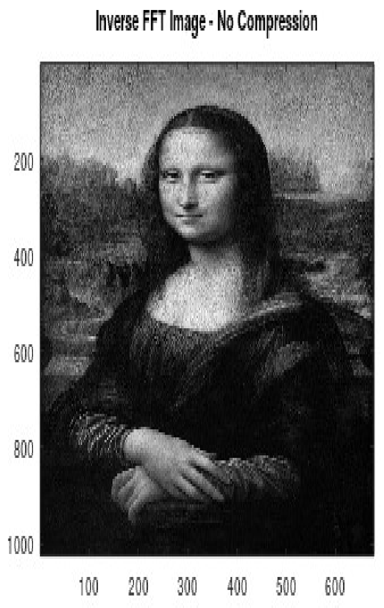
```
>>F = fft2(X);     (For given matrix X = [1,1,1,1; 1,1,1,1; 1,1,1,1; 1,1,1,1]
                    The transformed matrix is F = [16,0,0,0; 0,0,0,0; 0,0,0,0; 0,0,0,0] )
```

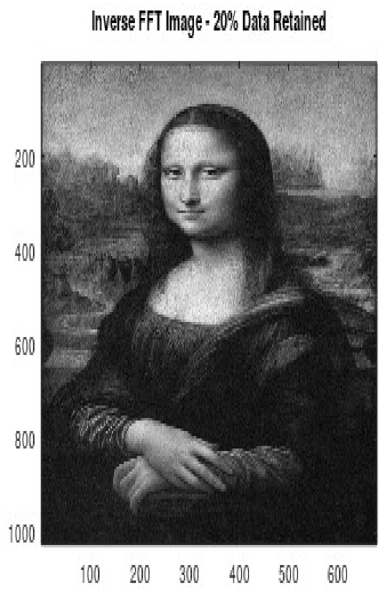
Octave has no specific command for data compression. A sample code is given below, for interested students! Note that transformed matrices have 'Complex Numbers'!!

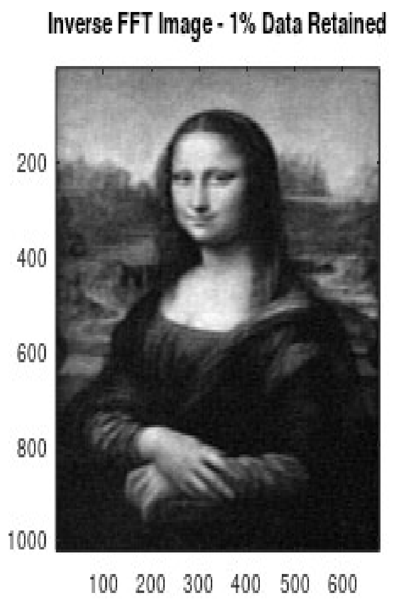
```
 function [f] = CompX(X,CompFact)
  Xsize = size(X);
  Xvec = abs(vec(X));                          # convert matrix to an array (vector) form!
  Xvec = sort(Xvec);                           # sort the array in ascending order

 CompIndex = uint32( XvecSize(1)*(1-CompFact) )  #  Get index based on Compression Factor
 Threshold = Xvec(CompIndex)                  # Get the Threshold value

 f = zeros(Xsize(1), Xsize(2));                # Create a matrix with all zeros

 for ix = 1:Xsize(1)                          # Retain only values >= 'Threshold'
   for jx = 1:Xsize(2)
     if ( abs(X(ix,jx)) >= abs(Threshold) ) f(ix,jx) = X(ix,jx);
     endif
   endfor
 endfor
endfunction
```

Octave Script for Image Processing using FFT

```
A = imread('Mona_Lisa1.jpg');       #  Get the colour image (jpg file) (A is a 3-d matrix)
colormap("gray");
B = rgb2gray(A);                    #  Convert to Black and White image  (B is a 2-d matrix)

BF = fft2(B);                       #  Calculate FFT of the image matrix  (BF)
BFInv = ifft2(BF);                  #  Calculate Inverse FFT to get back the image (BFInv)
BFComp = CompX(BF,0.2);             # CompX is the Function for Compression  (BFComp)
                                    # Compression Factor = 0.2 (20% retained)
BFCompInv = ifft2(BFComp);          # Calculate inverse FFT of the compressed FFT (BFCOMPInv)

figure 1;                           #  Display all images (figure 1 & figure 2)
subplot(2,1,1);
imagesc(A);
title("Original Colour Image");

subplot(2,1,2);
imagesc(B);
title("Original Image in Black & white");

figure 2;
colormap("gray");
subplot(2,1,1);
imagesc(real(BFInv));
title("Inverse FFT Image - No Compression");

subplot(2,1,2);
imagesc(real(BFCompInv));
title("Inverse FFT Image - 20% Data Retained");
```


Inverse FFT Image - No Compression
200
400
600
800
1000
100
200
300
400
500
600


Inverse FFT Image - 20% Data Retained
200
400
600
800
1000
100
200
300
400
500
600


Inverse FFT Image - 1% Data Retained
200
400
600
800
1000
100
200
300
400
500
600