**Batch: A-3 Roll No.: 16010122104**

**Experiment No. 3**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: C**ompute DFT & IDFT of discrete time signals using Matlab. |

**Objective:** To learn & understand the Fourier transform operations on discrete time signals.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO3** | Analyze signals in frequency domain through various image transforms |

**Books/ Journals/ Websites referred:**

1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html
3. www.mccormick.northwestern.edu/docs/efirst/matlab.pdf
4. A.Nagoor Kani “Digital Signal Processing”, 2nd Edition, TMH Education.

**Pre Lab/ Prior Concepts:**

**Implementation details along with screenshots:**

Given a sequence of *N* samples *f*(*n*), indexed by *n*= 0..*N*-1, the Discrete Fourier Transform (DFT) is defined as *F*(*k*), where *k*=0..*N*-1:

equation

*F*(*k*) are often called the 'Fourier Coefficients' or 'Harmonics'.

The sequence *f*(*n*) can be calculated from *F*(*k*) using the Inverse Discrete Fourier Transform (IDFT):

equation

In general, both *f*(*n*) and *F*(*k*) are complex.

Annex A shows that the IDFT defined above really is an *inverse* DFT.

Conventionally, the sequences *f*(*n*) and *F*(*k*) is referred to as 'time domain' data and 'frequency domain' data respectively. Of course there is no reason why the samples in *f*(*n*) need be samples of a time dependent signal. For example, they could be spatial image samples (though in such cases a 2 dimensional set would be more common).

Although we have stated that both *n* and *k* range over 0..*N*-1, the definitions above have a periodicity of *N*:

equation

So both *f*(*n*) and *F*(*k*) are defined for all (integral) *n* and *k* respectively, but we only need to calculate values in the range 0..*N*-1. Any other points can be obtained using the above periodicity property.

For the sake of simplicity, when considering various Fast Fourier Transform (FFT) algorithms, we shall ignore the scaling factors and simply define the FFT and Inverse FFT (IFFT) like this:

equation

equation

In fact, we shall only consider the FFT algorithms in detail. The inverse FFT (IFFT) is easily obtained from the FFT.

Here are some simple DFT's expressed as matrix multiplications.

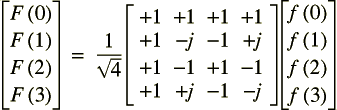
1 point DFT:

equation

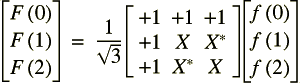
2 point DFT:

equation

4 point DFT:



3 point DFT:



equation

Note that each of the matrix multipliers can be inverted by conjugating the elements. This what we would expect, given that the only difference between the DFT and IDFT is the sign of the complex exponential argument.

Here's another couple of useful transforms:

If..

equation

equation

equation

This is the 'Delta Function'. The usual implied periodicity has been made explicit by using *MOD N*. The DFT is therefore:

equation

This gives us the DFT of a unit impulse at *n*=*n*0. Less obvious is this DFT:

If..

equation

equation

**Implementation steps with screenshots for DFT and IDFT:**

% IDFT Image Processing in MATLAB (Using fft2 for accuracy)

% Step 1: Input the size of the DFT matrix (must be a power of 2)

n = input('Enter the size of the DFT matrix (must be a power of 2): ');

% Check if the input size is a power of 2

if mod(log2(n), 1) ~= 0

error('The size of the DFT matrix must be a power of 2.');

end

% Step 2: Generate and Display the DFT matrix

dft\_matrix = zeros(n, n);

for k = 0:n-1

for l = 0:n-1

dft\_matrix(k+1, l+1) = exp(-2 \* pi \* 1i \* k \* l / n); % DFT formula

end

end

disp('DFT Matrix:');

disp(dft\_matrix);

% Step 3: Select and read the image

[filename, pathname] = uigetfile({'\*.jpg;\*.png;\*.bmp;\*.tif', 'Image Files (\*.jpg, \*.png, \*.bmp, \*.tif)'}, ...

'Select an image');

if filename == 0

error('No image selected. Please select an image.');

end

% Read the image and convert it to grayscale (if it is a color image)

image\_path = fullfile(pathname, filename);

image = imread(image\_path);

if size(image, 3) == 3

image = rgb2gray(image); % Convert to grayscale if the image is RGB

end

% Resize the image to fit the selected DFT size

image\_resized = imresize(image, [n, n]);

image\_double = double(image\_resized); % Convert to double for processing

% Step 4: Apply DFT using MATLAB’s fft2 function

dft\_image = fft2(image\_double);

% Step 5: Apply IDFT using MATLAB’s ifft2 function

idft\_image = ifft2(dft\_image); % This should reconstruct the original image

% Step 6: Ensure real values and convert to uint8

idft\_image = real(idft\_image); % Remove any small imaginary parts

idft\_image\_uint8 = uint8(idft\_image); % Convert back to uint8 (0-255)

% Step 7: Display the original, DFT magnitude, and IDFT images

figure;

subplot(1, 3, 1);

imshow(image\_resized, []);

title('Original Image');

subplot(1, 3, 2);

imshow(log(1 + abs(dft\_image)), []);

title('DFT Magnitude');

subplot(1, 3, 3);

imshow(idft\_image\_uint8, []);

title('Reconstructed Image (IDFT)');

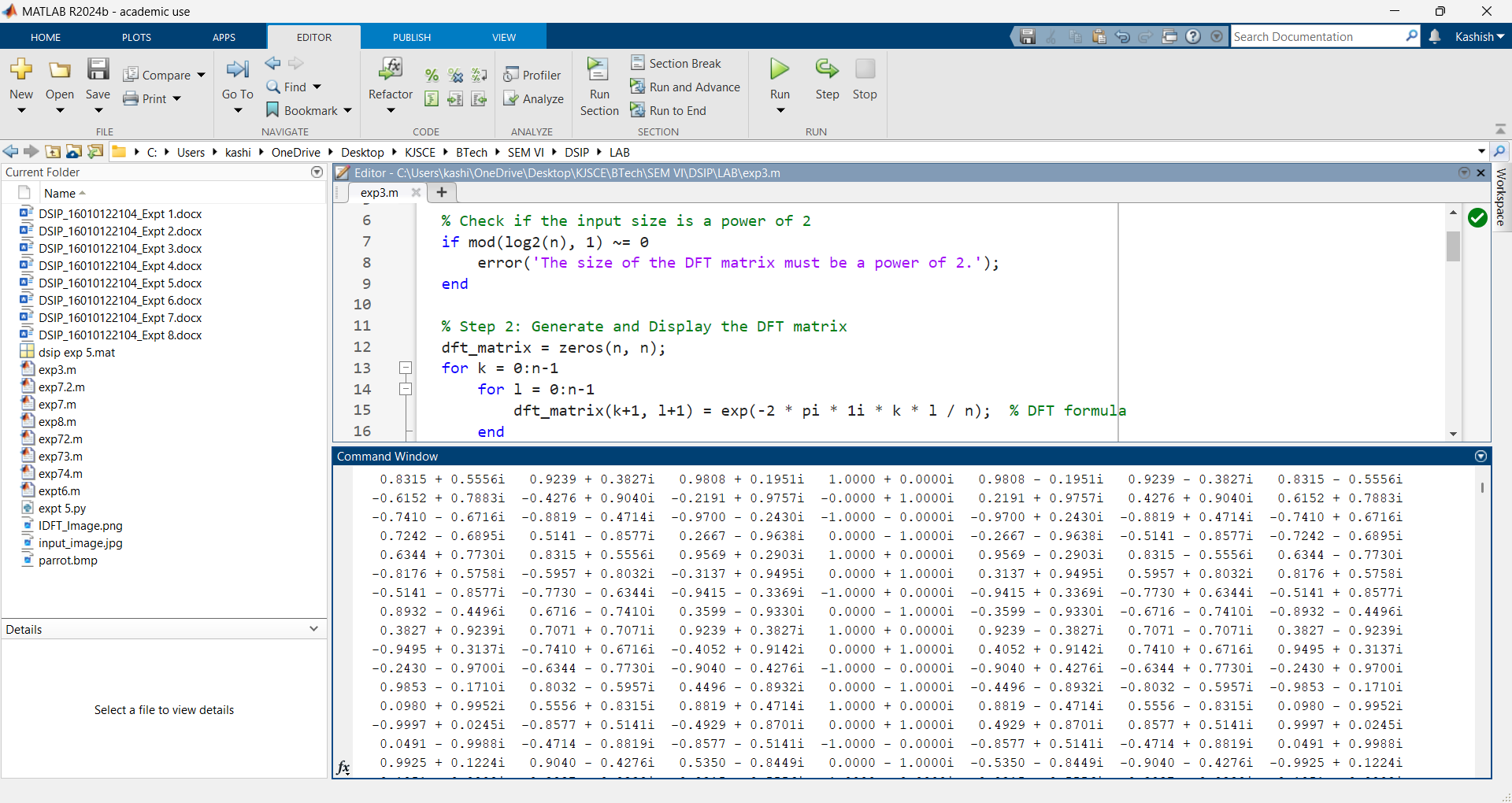
% Save the IDFT image

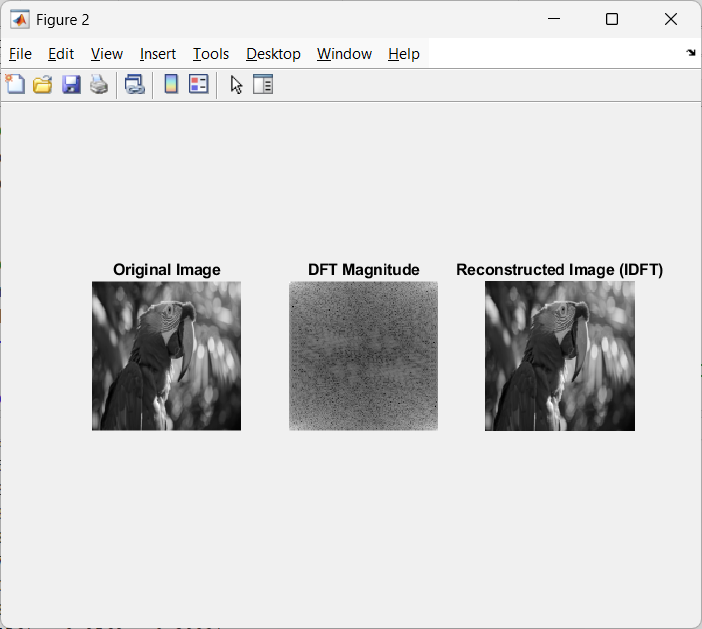
output\_filename = fullfile(pathname, 'IDFT\_Image.png');

imwrite(idft\_image\_uint8, output\_filename);

disp(['IDFT image saved as: ', output\_filename]);

**Output:**

****

****

****

**Conclusion:-**

This project successfully implemented DFT and IDFT in MATLAB to analyze images in the frequency domain and reconstruct them accurately. The results confirmed that IDFT restored the original image, demonstrating the effectiveness of Fourier Transform in image processing.

**Date: 08/04/2025 Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. **Compare and discuss the computational efficiency of DFT and FFT**

**Ans:**

Computational Efficiency: DFT vs. FFT

* The Discrete Fourier Transform (DFT) is defined mathematically and computed directly using its definition, which requires on the order of O(N2)*O*(*N*2) operations because every output sample is computed by summing N*N* products over N*N* input samples.
* In contrast, the Fast Fourier Transform (FFT) is a family of algorithms that compute the same transform in O(Nlog⁡N)*O*(*N*log*N*) time by exploiting properties such as symmetry and periodicity in the data.
* For large sequences, the reduction from quadratic to quasi-linear complexity means that the FFT is far more efficient, markedly reducing the number of multiplications and additions required.

1. **Give the properties of DFT and IDFT.**

**Ans:**

Properties of the DFT and IDFT

* **Linearity:** Both transforms are linear, meaning that the transform of a linear combination of signals equals the same combination of the transforms.
* **Time and Frequency Shifting:** A time-domain shift introduces a phase shift in the frequency domain, and vice versa; reversing the time order of a sequence reverses its frequency components.
* **Conjugation and Symmetry:** For real-valued signals, the DFT exhibits conjugate symmetry, meaning the imaginary parts of the frequency components appear in paired form, which is essential for signal reconstruction.
* **Orthogonality:** The complex exponential basis functions in the DFT are orthogonal over the interval of length N*N*, ensuring unique frequency representation.
* **Inversion Property:** The IDFT is defined as

x(n)=1N∑k=0N−1X(k)ej2πknN*x*(*n*)=*N*1*k*=0∑*N*−1*X*(*k*)*ejN*2*πkn*

and it exactly recovers the original time-domain sequence from its frequency-domain representation.

1. **Discuss the impact on computation time & efficiency when the number of samples N increases.**

**Ans:**

Impact of Increasing the Number of Samples

* In the direct DFT computation, the number of operations increases quadratically with N*N*; therefore, doubling the number of samples approximately quadruples the computation time.
* When using an FFT algorithm, on the other hand, the computational effort scales as O(Nlog⁡N)*O*(*N*log*N*), making it much more efficient as N*N* becomes large.
* This difference becomes especially significant in real-time or large-scale signal processing applications where N*N* is large, and computational efficiency is crucial.

1. **How to compute maximum length N for a circular convolution using DFT and IDFT?**

**Ans:**

Maximum Length N*N* for Circular Convolution Using DFT and IDFT

* Circular convolution computed via DFT and IDFT uses the periodicity inherent in the Fourier transform such that the output is periodic with period N*N*.
* To ensure that the circular convolution result matches the linear convolution (i.e., without aliasing), if two sequences have lengths L*L* and M*M*, then N*N* must be chosen so that

N≥L+M−1*N*≥*L*+*M*−1

This ensures that there is no overlap (aliasing) when wrapping the linear convolution result around the periodic domain.

* In practical implementations, zero-padding is applied to both sequences to the chosen length N*N* (often a power of two) to efficiently compute the convolution via the FFT while avoiding circular artifacts.

In summary, while the DFT computed by its definition scales poorly with sample size, the FFT offers a dramatic reduction in computation time for large N*N*. The inherent properties of both the DFT and the IDFT—linearity, shifting, conjugation, and orthogonality—support numerous signal processing operations, including circular convolution, for which proper selection of N*N* (typically N≥L+M−1*N*≥*L*+*M*−1) is critical to avoid aliasing.