EEE 424 Coding Assignment 1

In this assignment the aim was to compute and compare different Discrete Fourier Transform (DFT) methods.

Q1)

In this part of the assignment 5 different DFT approaches are implemented and applied to 3 different length arrays (N = 32, 256, 4096). Whole code of this question is provided in **Appendix A**.

These five different approaches are:

- DFT summation: Direct DFT Summation formula
- DFT matrix: DFT Matrix algorithm
- FFT-DIT: Fast Fourier Transform (FFT) using the decimation-in-time algorithm
- FFT-DIF: FFT using decimation-in-frequency algorithm
- fft: MATLAB’s built-in FFT command

Since the steps (a) to (h) are repeated for different lengths of arrays, steps (a)-(h) are written as a function called **steps_a_h(x)** which is provided in **Appendix A**.

First, a complex array of length N = 32 is created with the given code snippet:

```
rng(2,"twister")
N = 32;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
```

a)

Real and imaginary parts of x[n] of length N = 32 are plotted with the following code snippet:

```
n = 0:N-1;

figure;
subplot(2,1,1);
stem(n, real(x), 'filled');
title('Real Part of x[n]');
xlabel('n');
ylabel('Re\{x[n]\}');
xlim([0 N-1]);
grid on;

subplot(2,1,2);
stem(n, imag(x), 'filled');
title('Imaginary Part of x[n]');
xlabel('n');
ylabel('Im\{x[n]\}');
xlim([0 N-1]);
grid on;
```

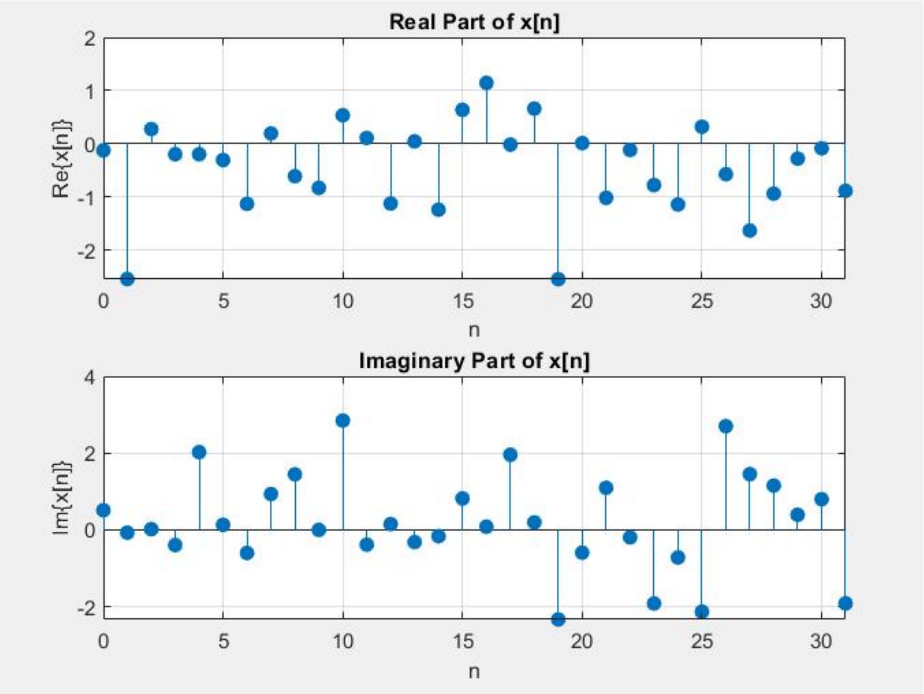

Fig.1 *Real and Imaginary parts of x[n], N= 32*

b)

The definition of DFT in summation form:

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j2\pi\frac{kn}{N}\right)$$

In order to compute N = 32 point DFT of the array x[n] using the direct definition in summation form the following function is defined:

```
function X = DFT_summation(x)
N = length(x);
X = zeros(1, N);
for k = 0:N-1
       for n = 0:N-1
              X(k+1) = X(k+1) + x(n+1) * exp(-1i * 2 * pi * k * n / N);
       end
end
end
```

In order to compute DFT of x[n], the function is called as follows:

```
t1 = tic;
Xsum = DFT_summation(x);
elapsed_time = toc(t1);
fprintf('Elapsed time for DFT_summation is %.6f seconds.\n', elapsed_time);
plotDFT(Xsum, 'Summation Formula')
```

c)

DFT matrix defined as:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

where:

$$\omega = e^{-j\frac{2\pi}{N}}$$

In order to create DFT matrix and compute the DFT of the array x[n] using the DFT matrix the following function is defined:

```
function X = DFT_matrix(x)
N = length(x);
WN = exp(-1i * 2 * pi / N);
DFT_mat = zeros(N, N);
for k = 0:N-1
      for n = 0:N-1
            DFT_mat(k+1, n+1) = WN^(k * n);
      end
end
X = (DFT_mat * x.').';
end
```

d)

DFT of x[n] calculated with the DFT matrix as follows:

$$X = \mathbf{W} \cdot \mathbf{x}$$

In order to compute DFT of x[n], the DFT matrix function is called as follows:

```
t2 = tic;
X_mat = DFT_matrix(x);
elapsed_time = toc(t2);
fprintf('Elapsed time for DFT_matrix is %.6f seconds.\n', elapsed_time);
plotDFT(X_mat, 'DFT matrix')
```

e)

Decimation-in-time Fast Fourier Transform is defined as recursively dividing the even and odd samples:

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn} = \sum_{n=0}^{N/2-1} x_{2n} W_N^{k(2n)} + \sum_{n=0}^{N/2-1} x_{2n+1} W_N^{k(2n+1)}$$

where:

$$W_N = e^{-i\frac{2\pi}{N}}$$

Key Steps in the FFT DIT Algorithm:

1. Divide: The sequence x is split into two smaller subsequences: one containing the even-indexed elements and the other containing the odd-indexed elements.

2. Conquer: The FFT of the even and odd subsequences is computed recursively.

3. Combine: The results of the even and odd subsequences are combined using a "twiddle factor" W, which is a complex exponential factor, to form the final result.

In order to compute the DFT of the array x[n] using the FFT decimation-in-time algorithm the following function is defined:

```
function X = FFT_DIT(x)
N = length(x);
if N == 1
       X = x;
else
       % Divide
       x_even = x(1:2:end);
       x_odd = x(2:2:end);
       % Conquer
       X_even = FFT_DIT(x_even);
       X_odd = FFT_DIT(x_odd);
       % Combine
       WN = exp(-1i * 2 * pi / N);
       W = 1;
       X = zeros(1, N);
       for k = 1:N/2
              X(k) = X_even(k) + W * X_odd(k);
              X(k + N/2) = X_even(k) - W * X_odd(k);
              W = W * WN;
       end
end
end
```

In order to compute DFT of x[n], the FFT DIT function is called as follows:

```
t3 = tic;
X_DIT = FFT_DIT(x);
elapsed_time = toc(t3);
fprintf('Elapsed time for FFT_DIT is %.6f seconds.\n', elapsed_time);
plotDFT(X_DIT, 'FFT-DIT')
```

f)

N point DFT defined as:

$$X[k] = \sum_{n=0}^{N-1} x_n W_N^{kn}$$

where:

$$W_N = e^{-i\frac{2\pi}{N}}$$

Decimation-in-frequency Fast Fourier Transform can be calculated by putting k = 2r in the summation formula then we get:

$$X[2r] = \sum_{n=0}^{N-1} x[n]W_N^{n(2r)}$$

Split into two sums:

$$X[2r] = \sum_{n=0}^{N/2-1} x[n]W_N^{2rn} + \sum_{n=0}^{N/2-1} x[n+N/2]W_N^{2r(n+N/2)}$$

Rearranging the terms:

$$X[2r] = \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2])W_{N/2}^{rn}$$

This N/2 DFT of first and second half summed. X[2r+1] can be found similarly.

This can be repeated recursively.

In order to compute the DFT of the array x[n] using the FFT decimation-in-frequency algorithm the following function is defined:

```
function X = FFT_DIF(x)
N = length(x);
if N == 1
       X = x;
else
       X = x;
       % Butterfly stage
       for k = 1:N/2
             temp = X(k);
             X(k) = temp + X(k + N/2);
             X(k + N/2) = (temp - X(k + N/2)) * exp(-1i * 2 * pi * (k - 1) / N);
       end
       % Recursive stage
       X(1:N/2) = FFT_DIF(X(1:N/2));
       X(N/2+1:N) = FFT_DIF(X(N/2+1:N));
end
end
```

In order to compute DFT of x[n], the FFT DIF function is called as follows:

```
t4 = tic;
X_DIF = bitrevorder(FFT_DIF(x));
elapsed_time = toc(t4);
fprintf('Elapsed time for FFT_DIF is %.6f seconds.\n', elapsed_time);
plotDFT(X_DIF, 'FFT-DIF')
```

g)

In order to compute DFT of x[n] using the MATLAB's built-in FFT command, the fft command is called as follows:

```
t5 = tic;
X_fft = fft(x);
elapsed_time = toc(t5);
fprintf('Elapsed time for fft is %.6f seconds.\n', elapsed_time);
plotDFT(X_fft, 'FFT')
```

h)

For plotting purposes, **plotDFT(X, str)** function is defined. That takes the name of the algorithm (str) and DFT of x[n] (X[k]) and plots the magnitude and phase graphs of X[k]. The function is visible in **Appendix A**.
The magnitude and phase graphs of X[k] for each algorithm when N = 32 are below in Figures 2-6:

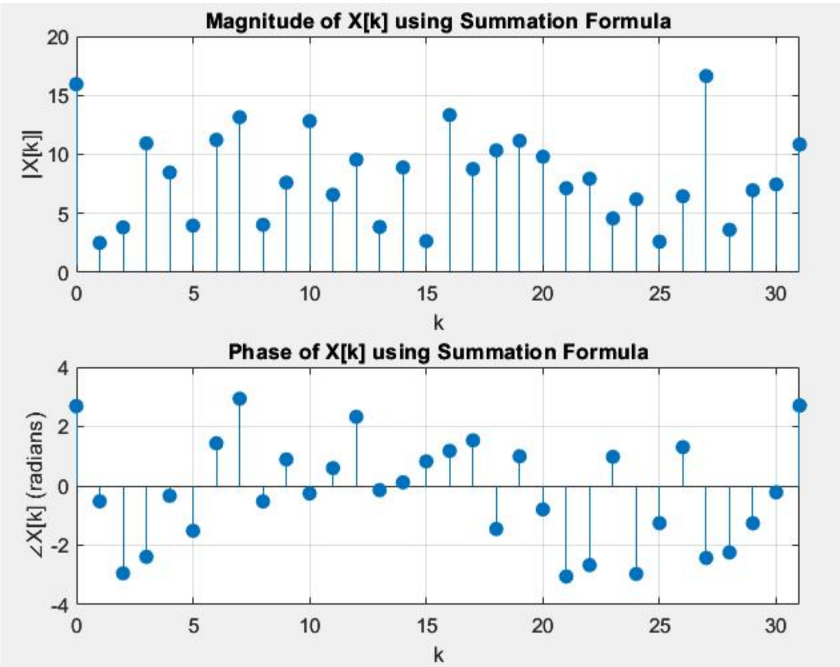

Fig.2 *DFT using summation formula, N = 32*

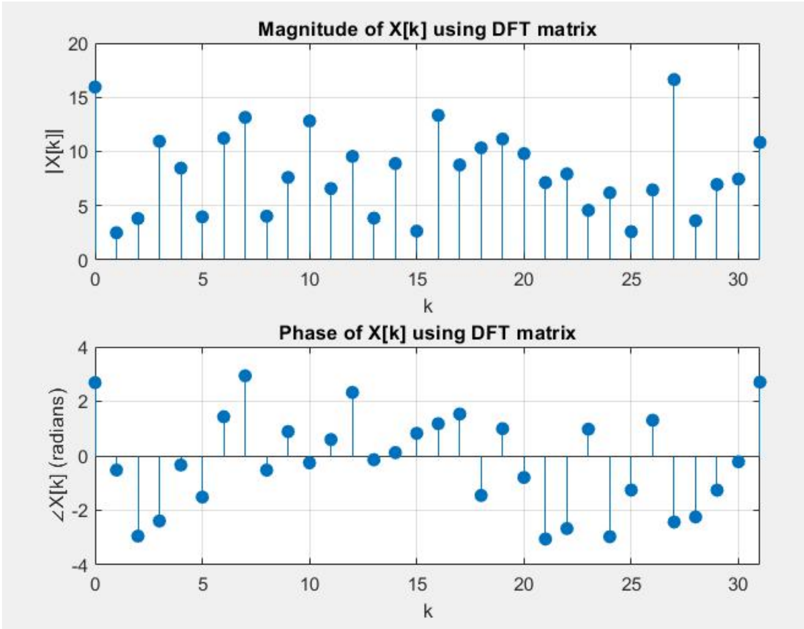

Fig.3 *DFT using DFT matrix, N = 32*

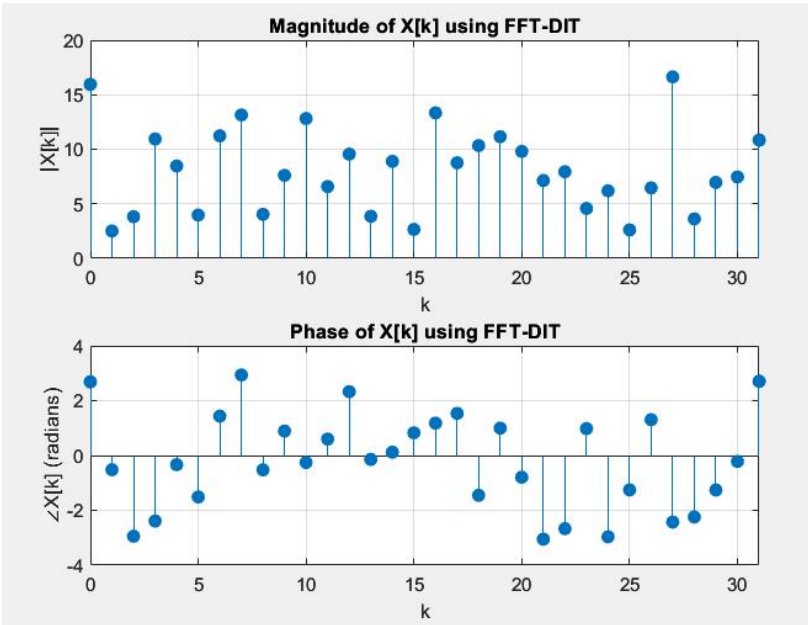

Fig.4 *DFT using FFT-DIT, N = 32*

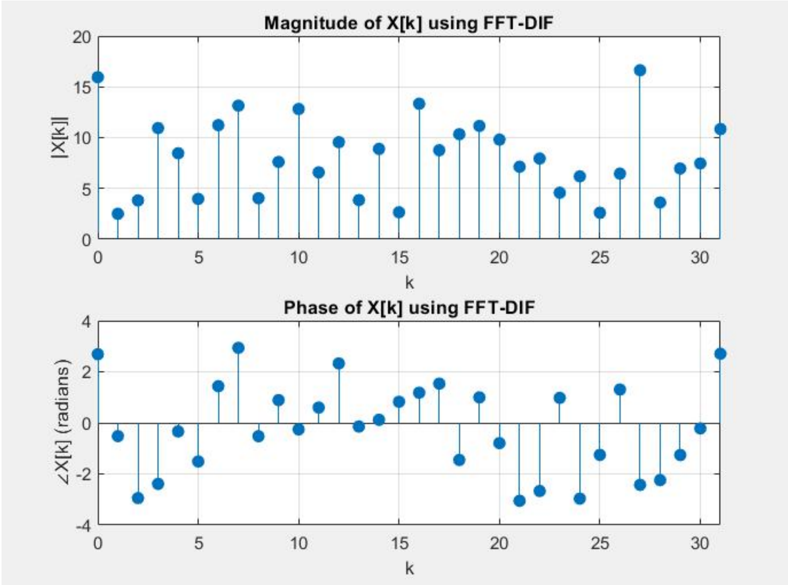

***Fig.5** DFT using FFT-DIF, N = 32*

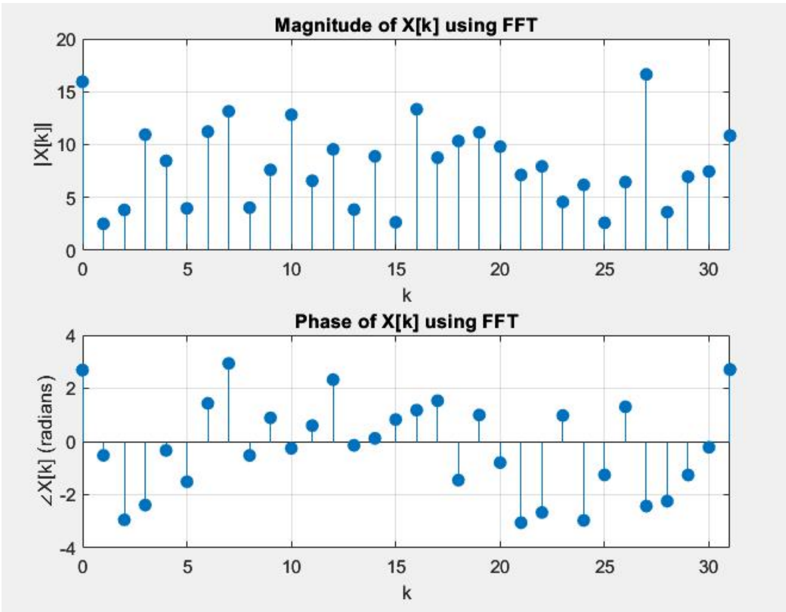

***Fig.6** DFT using FFT, N = 32*

Comparing Figures 2-6 it is visible that each algorithm produces equivalent results.

The algorithms are also compared using the difference of the norms between arrays. Each custom algorithm is compared with the built-in FFT command to check whether the norm is infinitesimally small or not (close to machine precision). The following code snippet computes the differences:

```
disp(['Summation Formula: ', num2str(norm(X_fft - Xsum))]);
disp(['DFT matrix: ', num2str(norm(X_fft - X_mat))]);
disp(['FFT-DIT: ', num2str(norm(X_fft - X_DIT))]);
disp(['FFT-DIF: ', num2str(norm(X_fft - X_DIF))]);
disp(['fft: ', num2str(norm(X_fft - X_fft))]);
```

And the output is as following:

```
Summation Formula:   3.1489e-13
DFT matrix:          4.8211e-13
FFT-DIT:             1.4408e-14
FFT-DIF:             1.2255e-14
fft:                 0
```

Fig.7 *Difference of the norms between the algorithms, N = 32*

From Fig.7 it is evident that the difference of the norms are very close to zero, therefore, all the algorithms generates the same result.

i)

Now, steps (a) - (h) repeated for a array of length with N = 256. The vector is crated using the provided code snippet as follows:

```
rng(2,"twister")
N = 256;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
```

And the steps (a)-(h) are repeated using the above mentioned custom function **steps_a_h(x)** (see Appendix A):

```
steps_a_h(x);
```

Real and imaginary parts of x[n] of length N = 256 are plotted:

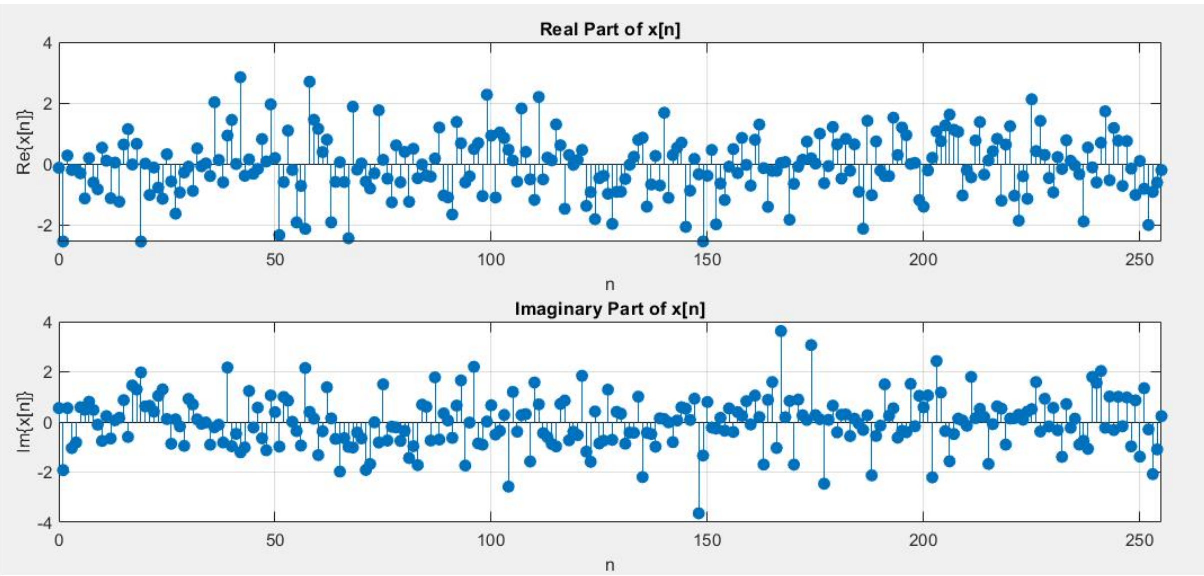

Fig.8 *Real and Imaginary parts of x[n], N= 256*

The magnitude and phase graph of the different algorithms:

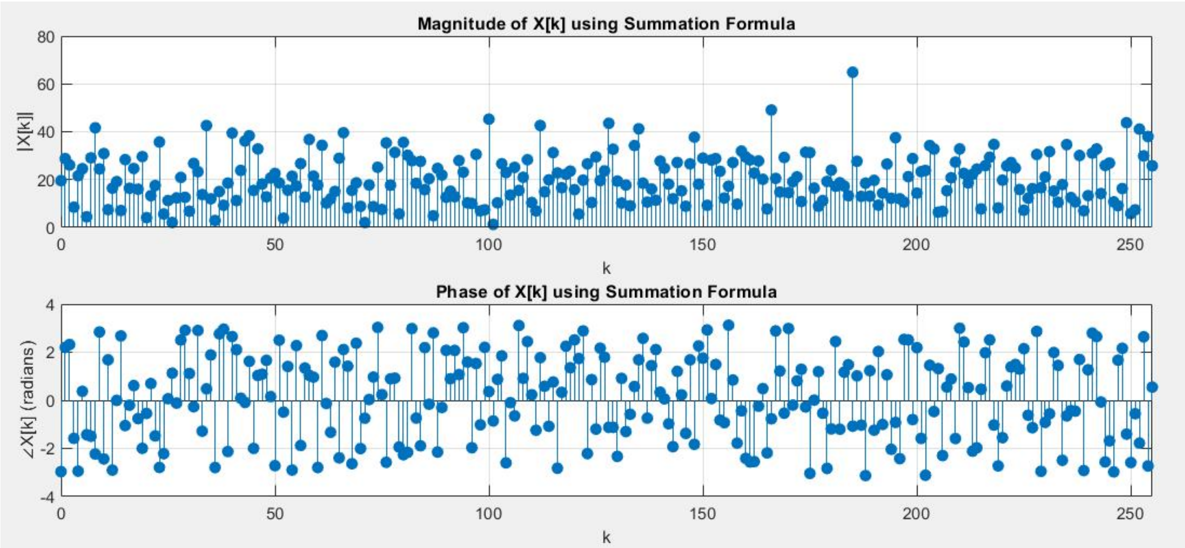

Fig.9 *DFT using summation formula, N = 256*

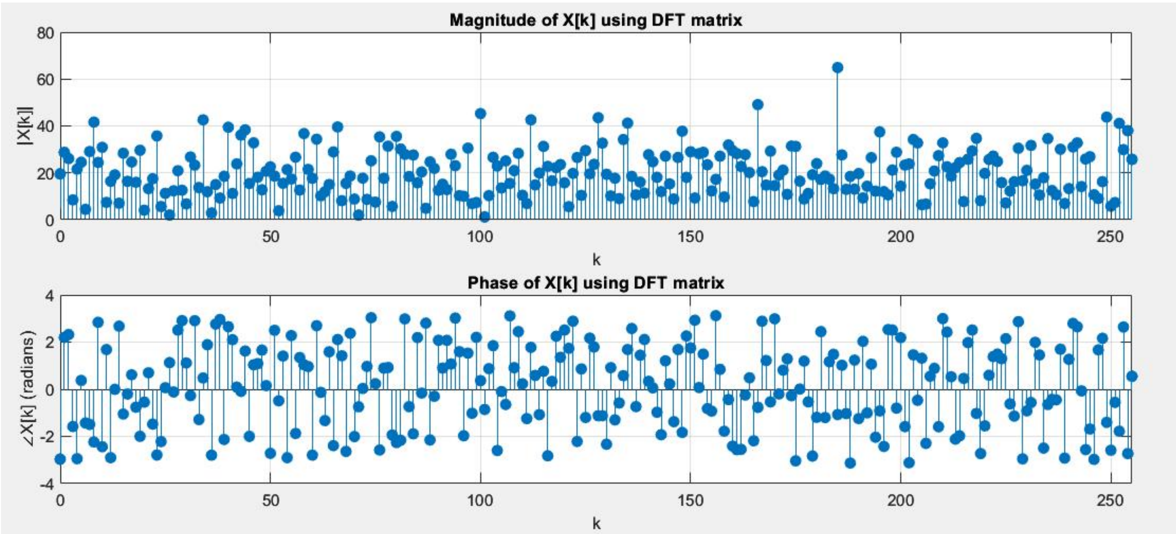

Fig.10 *DFT using DFT matrix, N = 256*

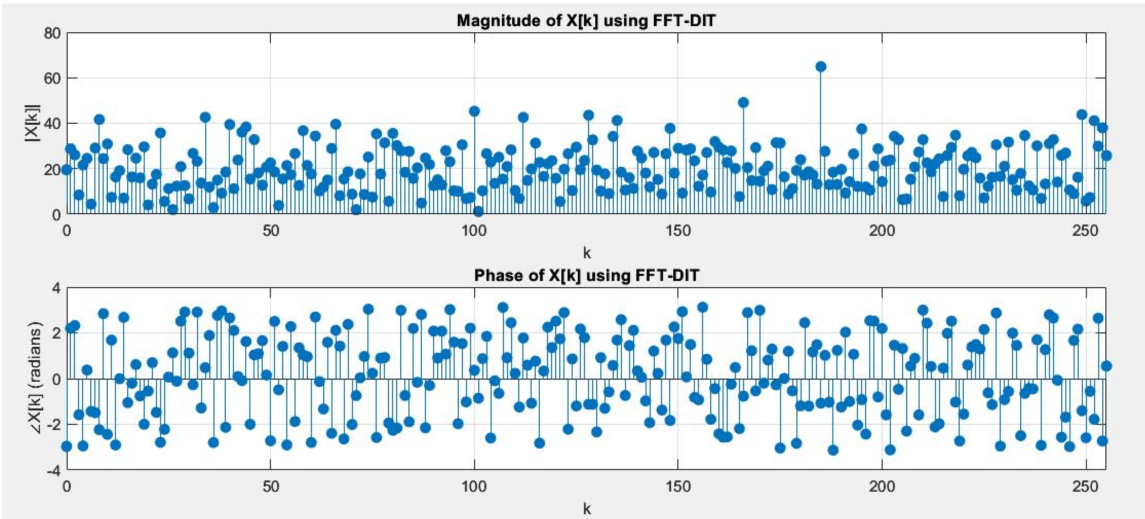

Fig.11 *DFT using FFT-DIT, N = 256*

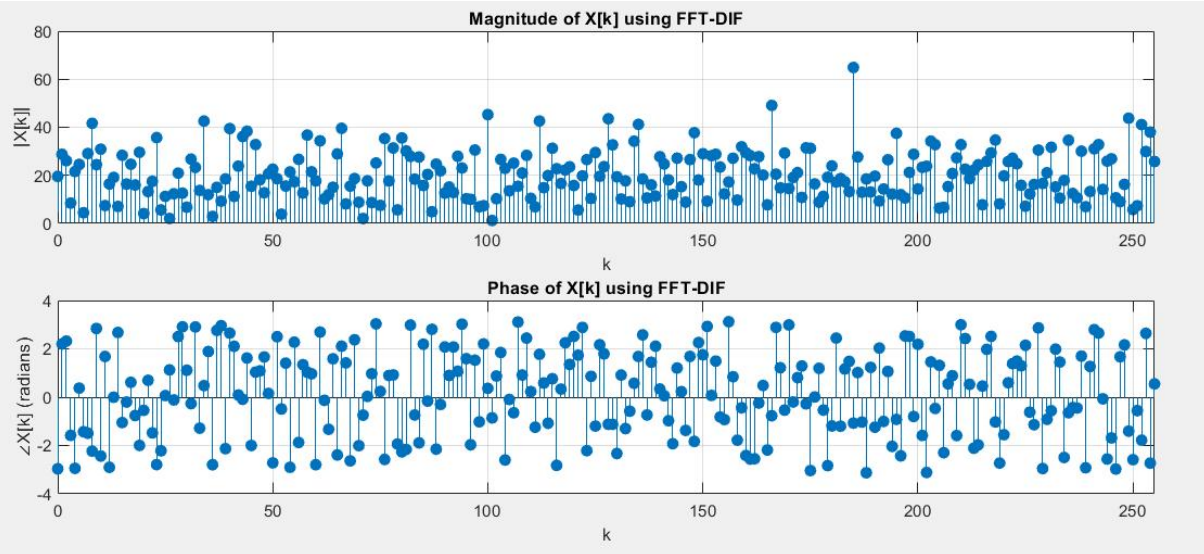

Fig.12 *DFT using FFT-DIF, N = 256*

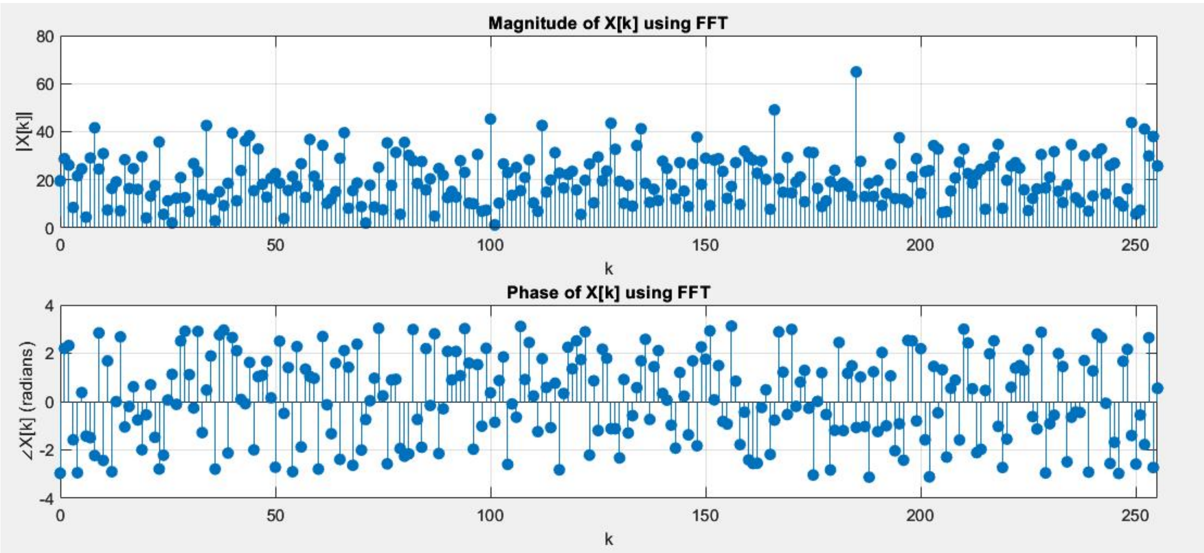

Fig.13 *DFT using FFT, N = 256*

Comparing Figures 9-13 it is visible that each algorithm produces equivalent results.

Differences of the norms of different algorithms:

```
Summation Formula:   1.5703e-11
DFT matrix:          1.1515e-10
FFT-DIT:             6.6354e-13
FFT-DIF:             1.4041e-13
fft:                 0
```

Fig.14 *Difference of the norms between the algorithms, N = 256*

From Fig.14 it is evident that the difference of the norms are very close to zero, therefore, all the algorithms generates the same result.

j)

Now, steps (a) - (h) repeated for a array of length with N = 2^12. The vector is crated using the provided code snippet as follows:

```
rng(2,"twister")
N = 2^12;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
```

And the steps (a)-(h) are repeated using the above mentioned custom function **steps_a_h(x)** (see Appendix A):

```
steps_a_h(x);
```

Real and imaginary parts of x[n] of length N = 2^12 are plotted:

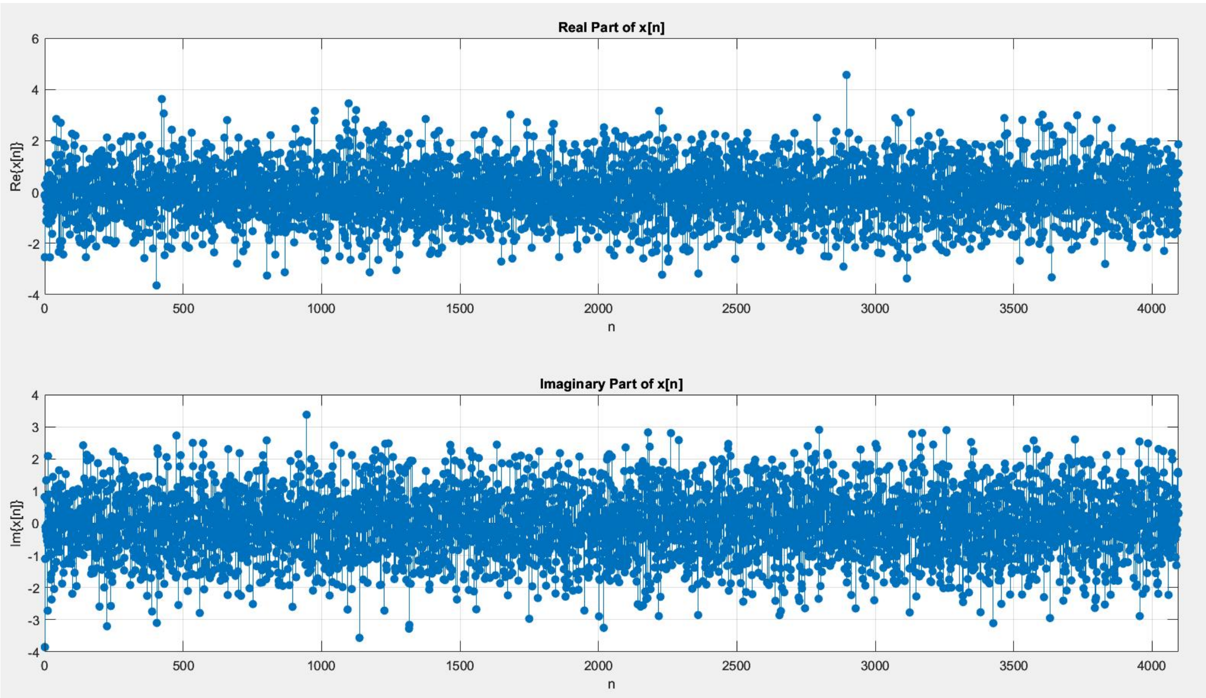

Fig.15 *Real and Imaginary parts of x[n], N=* 2^12

The magnitude and phase graph of the different algorithms:

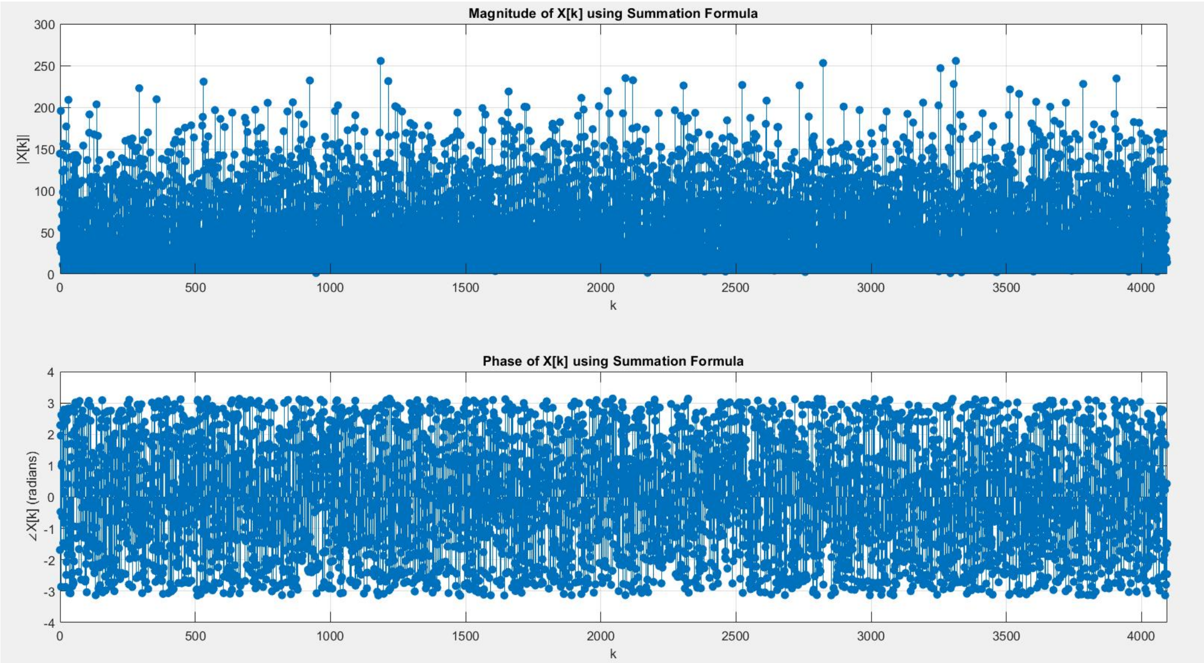

Fig.16 *DFT using summation formula, N* = 2^12

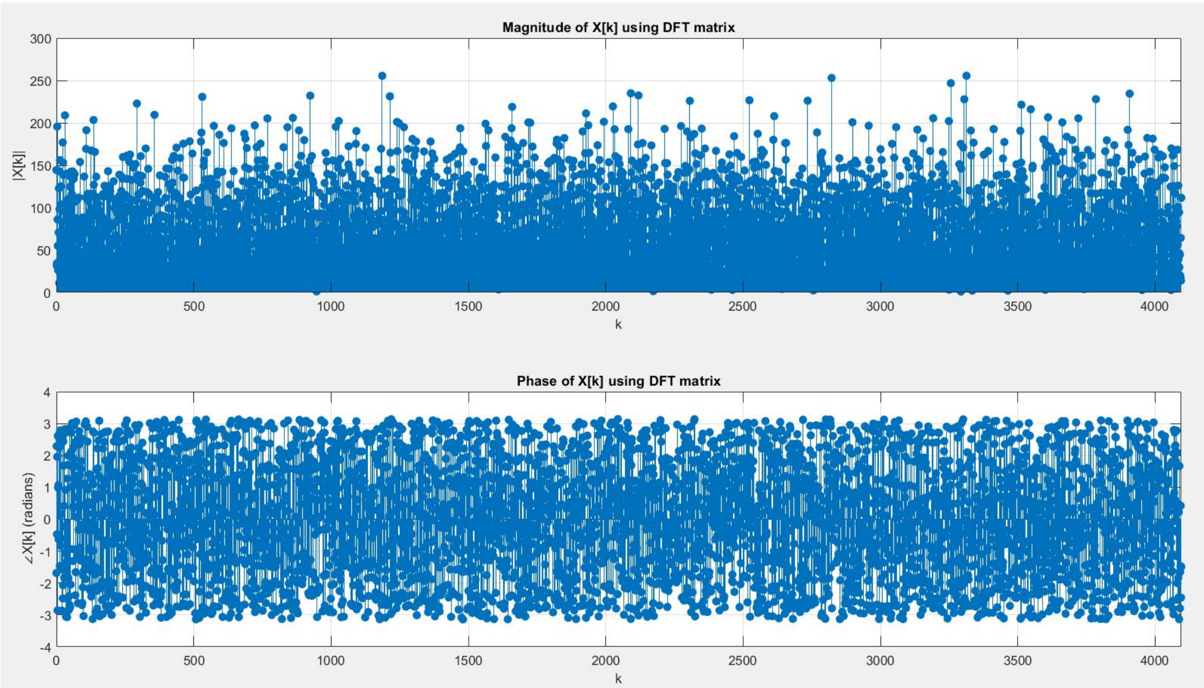

Fig.17*DFT using DFT matrix, N* = 2^12

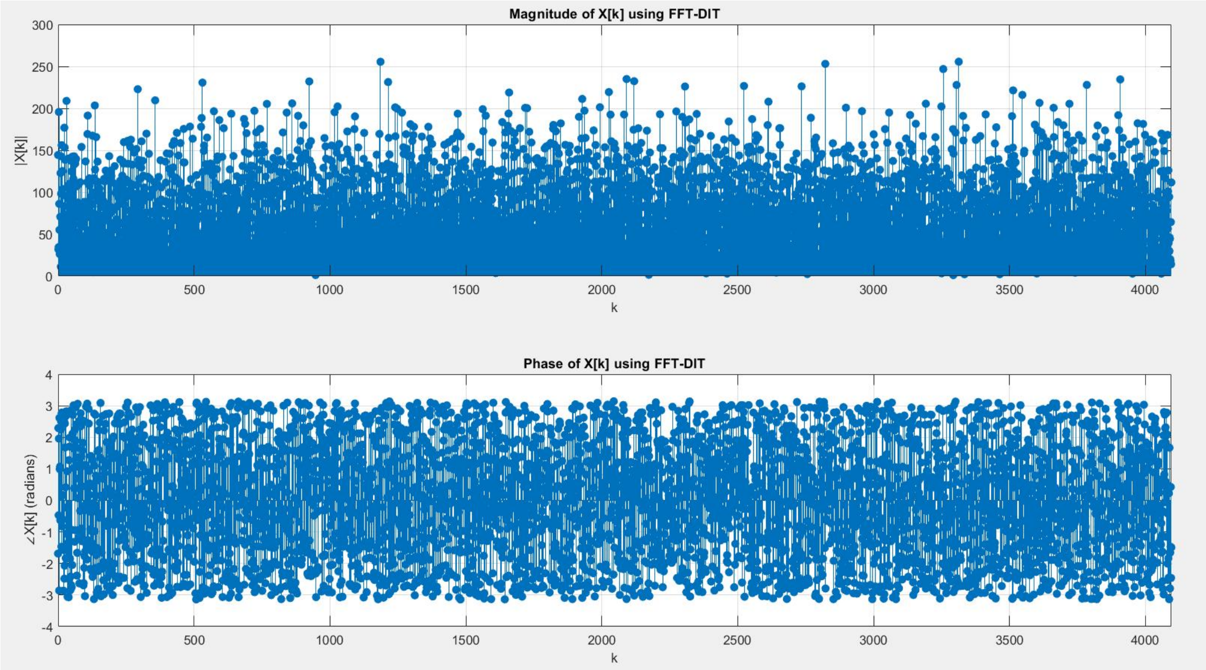

Fig.18 *DFT using FFT-DIT, N* = 2^12

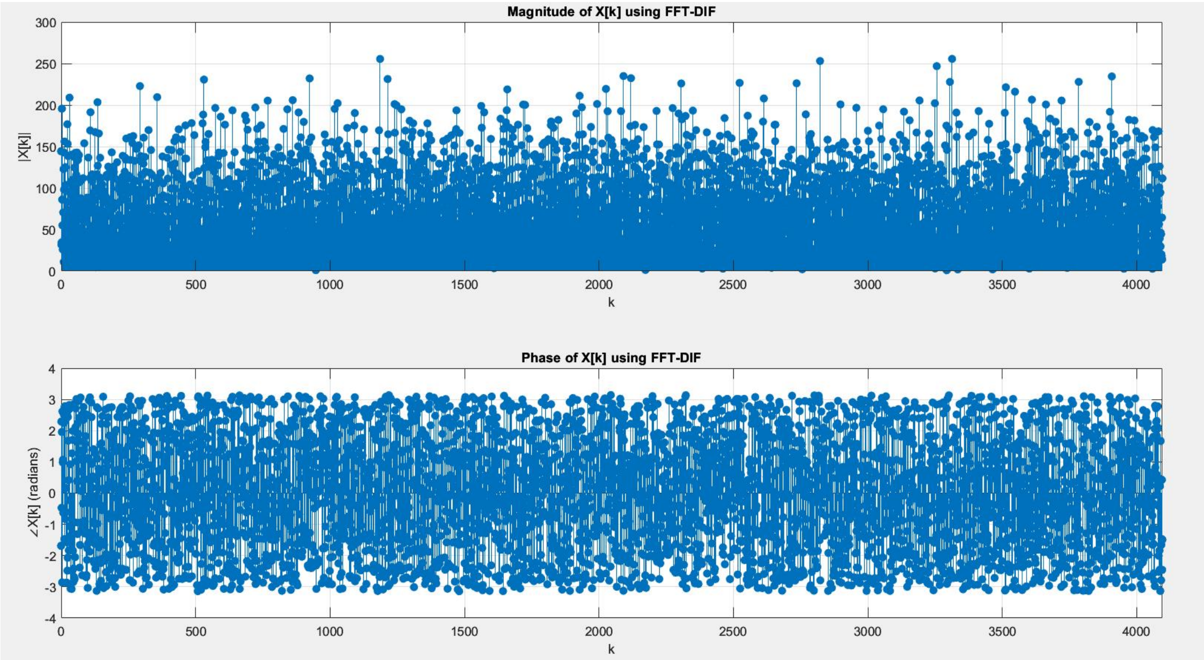

Fig.19 *DFT using FFT-DIF, N* = 2^12

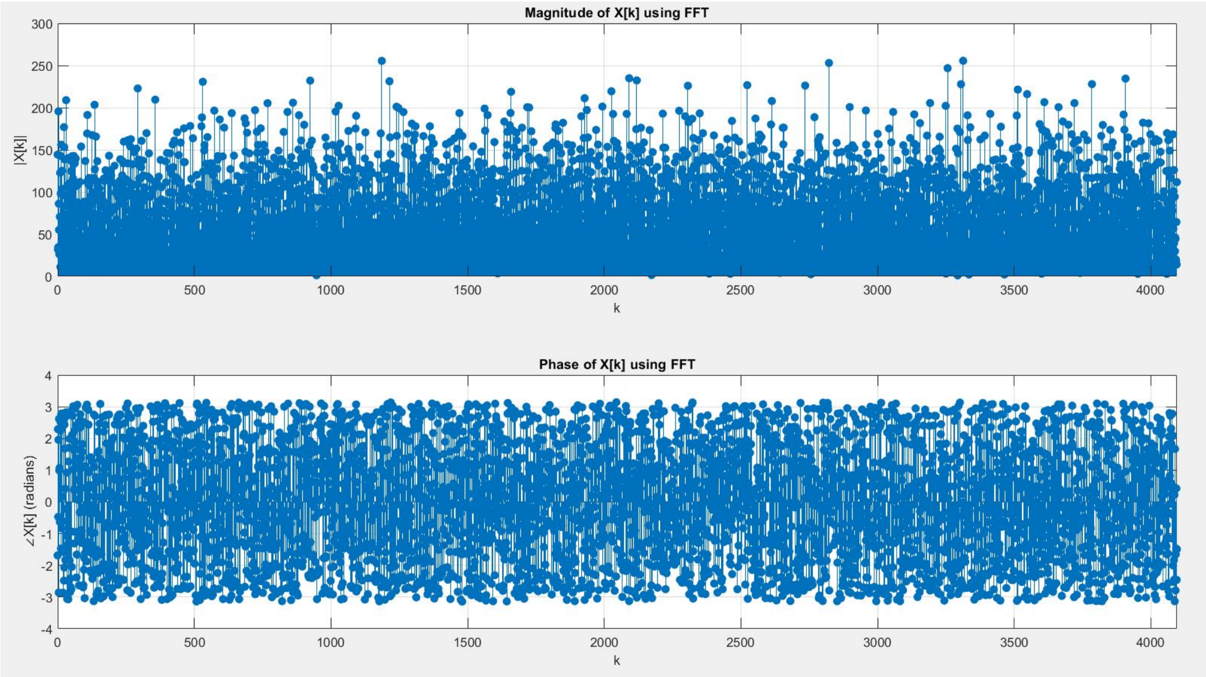

Fig.20 *DFT using FFT, N* = 2^12

Comparing Figures 16-20 it is visible that each algorithm produces equivalent results.

Differences of the norms of different algorithms:

```
Summation Formula:    3.9066e-09
DFT matrix:           1.1446e-06
FFT-DIT:              1.7455e-10
FFT-DIF:              3.0277e-12
fft:                  0
```

Fig.21 *Difference of the norms between the algorithms, N* = 2^12

From Fig.21 it is evident that the difference of the norms are very close to zero, therefore, all the algorithms generates the same result.

k)

Time it takes for each algorithm is calculated using *tic* and *toc* commands. The results are gathered in the Table 1.

	N = 32	N = 256	N = 2^12
Part (b)	0.001658 s	0.018429 s	4.120413 s
Part (d)	0.001597 s	0.034487 s	8.044837 s
Part (e)	0.000797 s	0.004292 s	0.019670 s
Part (f)	0.016621 s	0.004007 s	0.032825 s
Part (g)	0.000464 s	0.002475 s	0.000225 s

Table 1: *Time it takes to compute different DFT approaches on signals of varying length*

Table 1 shows the time taken by five different DFT approaches for three signal lengths (N = 32, 256, and 4096). As expected, the time increases with larger signal sizes, and there is a significant variation in time across different methods. The direct DFT summation (part b) and DFT matrix (part d) exhibit a much slower computation time for larger arrays compared to the FFT-based approaches. Specifically, the FFT-DIT (part e) and FFT-DIF (part f) methods perform significantly faster as the signal length increases, highlighting the advantages of these algorithms in handling large data sets. The built-in MATLAB FFT (part g) shows the fastest computation time, aligning with theoretical expectations that FFT-based methods reduce complexity from O(N^2) in the direct summation to O(NlogN). These findings confirm the efficiency of FFT algorithms, particularly as the signal length grows.

Q2)

In this part, we are focusing on a uncommon scenario where we are computing FFT of a signal length $N = 3^v$ instead of $N = 2^v$. In particular, 9-pt signal array. Whole code of this question is provided in **Appendix B**.

a)

Following similar steps with part (f) where we computed FFT of a signal with length $N = 2^v$ using the decimation-in-frequency algorithm.
N point DFT defined as:

$$X[k] = \sum_{n=0}^{N-1} x_n W_N^{kn}$$

where:

$$W_N = e^{-i\frac{2\pi}{N}}$$

So in order to find X[k], let k = 3r, where r = 0 ,…, N/3-1

$$X[3r] = \sum_{n=0}^{N-1} x_n W_N^{(3r)n}$$

Split this over 3 sums with N/3 intervals.

$$X[3r] = \sum_{n=0}^{N/3-1} x[n]W_N^{3rn} + \sum_{n=0}^{N/3-1} x[n+N/3]W_N^{3r(n+N/3)} + \sum_{n=0}^{N/3-1} x[n+2N/3]W_N^{3r(n+2N/3)}$$

$$W_N^{3rn} = W_{N/3}^{rn}$$

$$W_N^{3r(n+N/3)} = W_N^{3rn} \cdot W_N^{Nr} = W_{N/3}^{rn} \quad , \; W_N^{Nr} = 1$$

$$W_N^{3r(n+2N/3)} = W_N^{3rn} \cdot W_N^{2Nr} = W_{N/3}^{rn} \quad , \; W_N^{2Nr} = 1$$

$$X[3r] = \sum_{n=0}^{N/3-1} (x[n] + x[n+N/3] + x[n+2N/3])W_{N/3}^{rn}$$

For k = 3r +1 where r = 0 ,…, N/3-1

$$X[3r+1] = \sum_{n=0}^{N-1} x_n W_N^{(3r+1)n}$$

Split this over 3 sums with N/3 intervals.

$$X[3r+1] = \sum_{n=0}^{N/3-1} x[n]W_N^{(3r+1)n} + \sum_{n=0}^{N/3-1} x[n+N/3]W_N^{(3r+1)(n+N/3)} + \sum_{n=0}^{N/3-1} x[n+2N/3]W_N^{(3r+1)(n+2N/3)}$$

$$W_N^{(3r+1)n} = W_{N/3}^{rn} \cdot W_N^{n}$$

$$W_N^{(3r+1)(n+N/3)} = W_N^{3rn} \cdot W_N^{n} \cdot W_N^{Nr} \cdot W_N^{N/3} = W_{N/3}^{rn} \cdot W_N^{n} \cdot W_N^{N/3} \quad , \ W_N^{Nr} = 1$$

$$W_N^{(3r+1)(n+2N/3)} = W_N^{3rn} \cdot W_N^{n} \cdot W_N^{2Nr} \cdot W_N^{2N/3} = W_{N/3}^{rn} \cdot W_N^{n} \cdot W_N^{2N/3} \quad , \ W_N^{2Nr} = 1$$

$$X[3r+1] = \sum_{n=0}^{N/3-1} (x[n] + x[n+N/3]W_N^{N/3} + x[n+2N/3]W_N^{2N/3})W_{N/3}^{rn} \cdot W_N^{n}$$

For k = 3r +2 where r = 0 ,…, N/3-1

$$X[3r+2] = \sum_{n=0}^{N-1} x_n W_N^{(3r+2)n}$$

Split this over 3 sums with N/3 intervals.

$$X[3r+2] = \sum_{n=0}^{N/3-1} x[n]W_N^{(3r+2)n} + \sum_{n=0}^{N/3-1} x[n+N/3]W_N^{(3r+2)(n+N/3)} + \sum_{n=0}^{N/3-1} x[n+2N/3]W_N^{(3r+2)(n+2N/3)}$$

$$W_N^{(3r+2)n} = W_{N/3}^{rn} \cdot W_{N/2}^{n}$$

$$W_N^{(3r+2)(n+N/3)} = W_N^{3rn} \cdot W_{N/2}^{n} \cdot W_N^{Nr} \cdot W_N^{2N/3} = W_{N/3}^{rn} \cdot W_N^{n} \cdot W_N^{2N/3} \quad , \ W_N^{Nr} = 1$$

$$W_N^{(3r+2)(n+2N/3)} = W_N^{3rn} \cdot W_N^{2n} \cdot W_N^{2Nr} \cdot W_N^{4N/3} = W_{N/3}^{rn} \cdot W_{N/2}^{n} \cdot W_N^{4N/3} \quad , \ W_N^{2Nr} = 1$$

$$X[3r+2] = \sum_{n=0}^{N/3-1} (x[n] + x[n+N/3]W_N^{2N/3} + x[n+2N/3]W_N^{4N/3})W_{N/3}^{rn} \cdot W_N^{2n}$$

b)

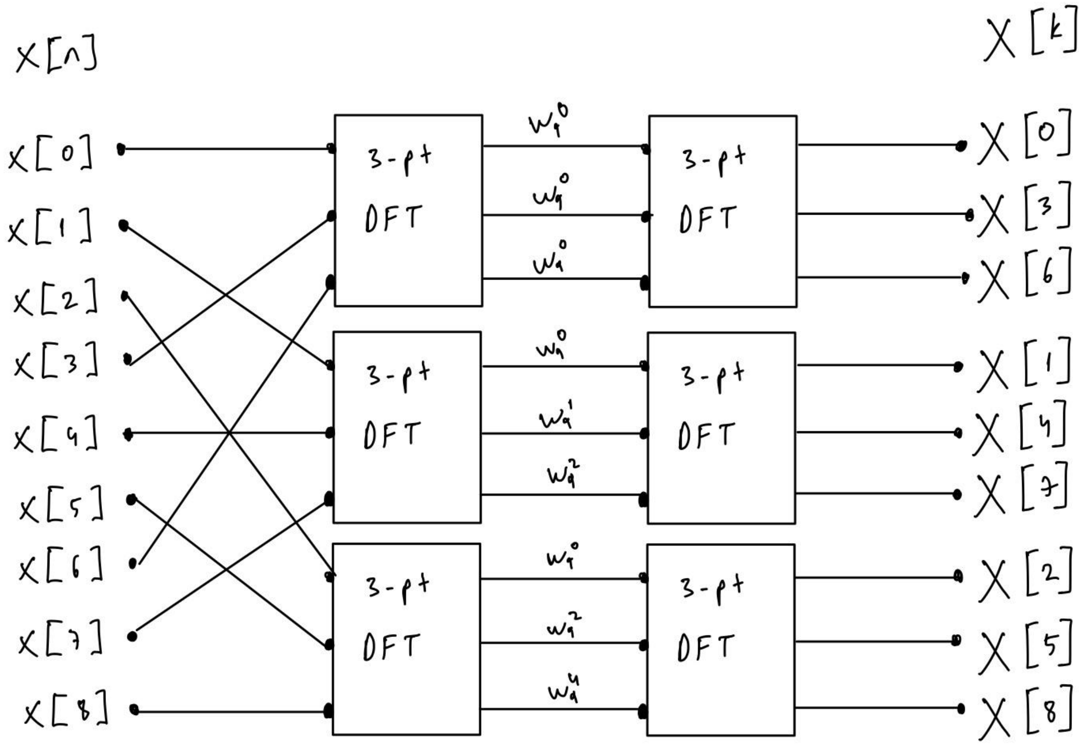

Fig.21 *Sample flow graph for the 9-pt decimation-in-frequency FFT algorithm.*

c)

First, a complex array of length N = 9 is created with the given code snippet:

```
rng(2,"twister")
N = 9;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
```

Real and imaginary parts of x[n] are plotted:

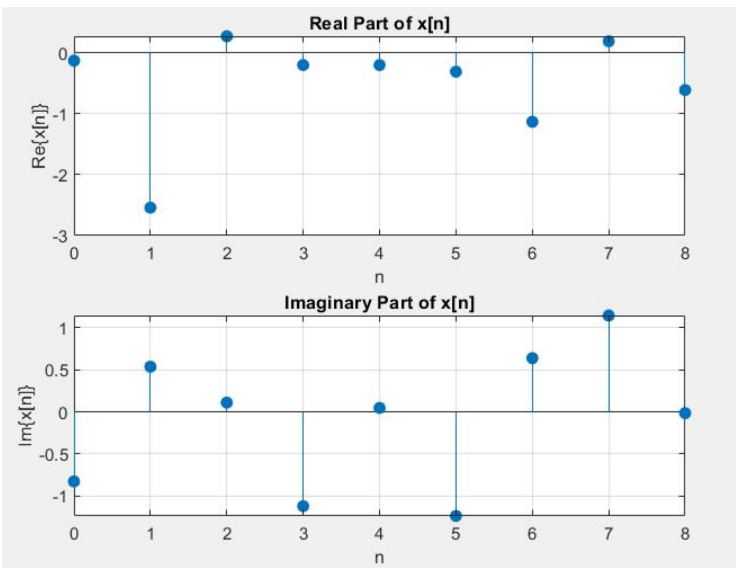

Fig.22 *Real and Imaginary parts of x[n], N= 9*

In order to compute DFT of array x using the DFT definition in summation form, the same function defined in ***Q1)*** *(b)*, **DFT_summation(x)**, has been called as in the following code snippet:

```
t1 = tic;
Xsum = DFT_summation(x);
elapsed_time = toc(t1);
fprintf('Elapsed time for DFT_summation is %.6f seconds.\n', elapsed_time);
plotDFT(Xsum, 'Summation Formula')
```

d)

In order to implment the 9-pt FFT using decimation-in-frequency algorithm derived in part (a) the following function is defined:

```
function X = nineptFFT(x)
N = length(x);
X = zeros(1, N);
for r = 0:(N/3 - 1)
for p = 0:2
temp = 0;
for n = 0:(N/3 - 1)
inner_sum = 0;
for l = 0:2
inner_sum = inner_sum + x(n + l*N/3 + 1) * exp(-1j*2*pi*p*l/3);
end
temp = temp + inner_sum * exp(-1j*2*pi*n*r/(N/3)) * exp(-1j*2*pi*p*n/N);
end
X(3*r + p + 1) = temp;
end
end
end
```

The DFT of x[n] found with the following code snippet:

```
t2 = tic;
X_9pt_fft = nineptFFT(x);
elapsed_time = toc(t2);
fprintf('Elapsed time for 9-pt FFT is %.6f seconds.\n', elapsed_time);
plotDFT(X_9pt_fft, '9-pt FFT')
```

e)

In order to compute DFT of x[n] using the MATLAB's built-in FFT command, the fft command is called as follows:

```
t3 = tic;
X_fft = fft(x);
elapsed_time = toc(t3);
fprintf('Elapsed time for fft is %.6f seconds.\n', elapsed_time);
plotDFT(X_fft, 'FFT')
```

f)

In order to compare the results norm difference of the algorithms are computed following the same steps in ***Q1)*** *(h)*.

The magnitude and phase graph of the different algorithms:

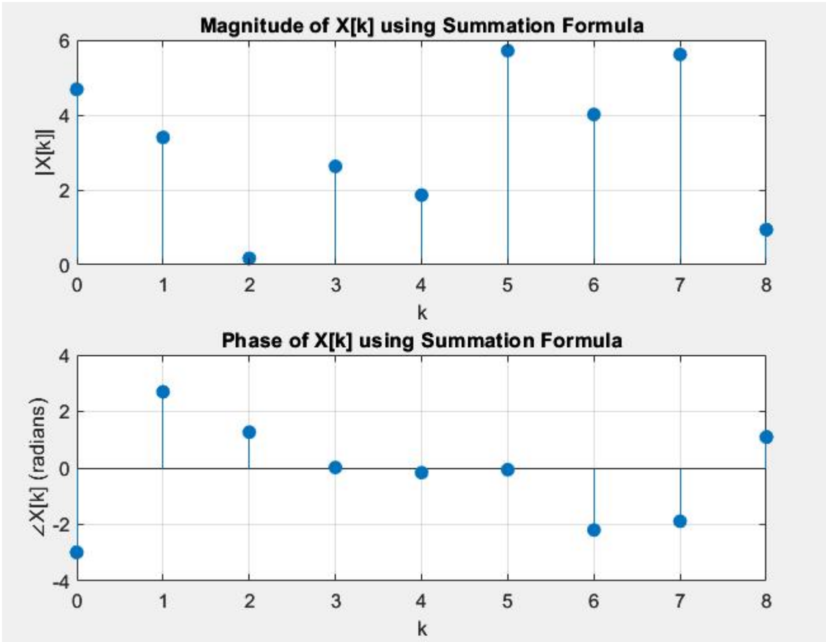

Fig.23 *DFT using summation formula, N = 9*

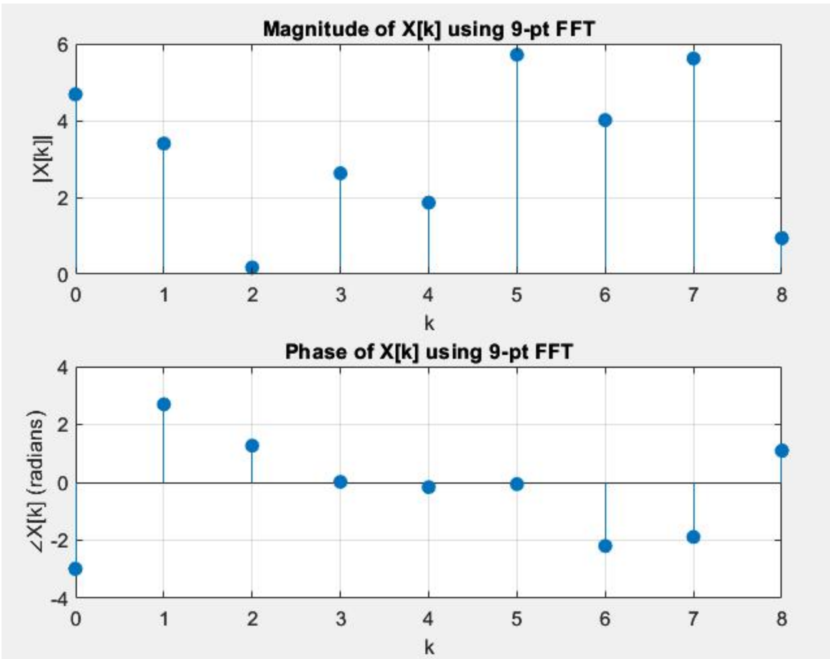

Fig.24 *DFT using 9-pt FFT DIF, N = 9*

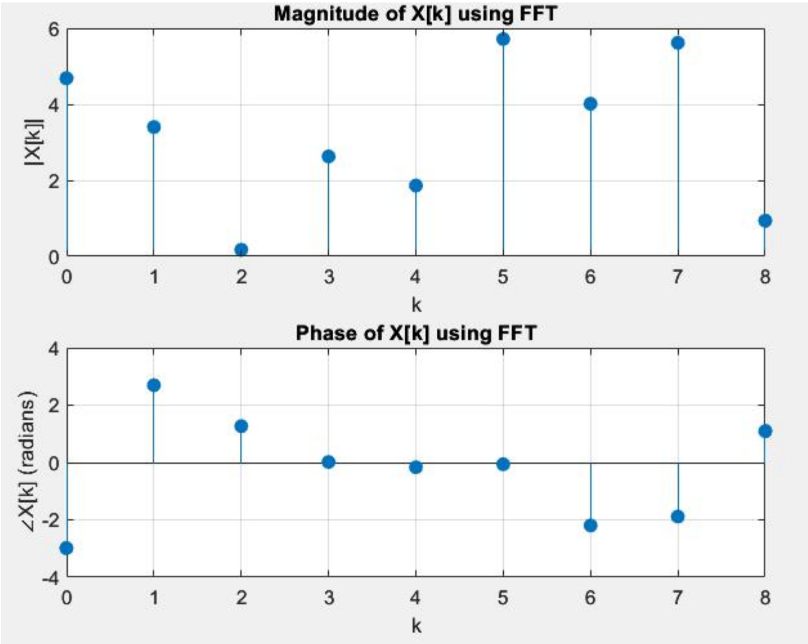

Fig.25 DFT using FFT, N = 9

Comparing Figures 23-25 it is visible that each algorithm produces equivalent results.

Differences of the norms of different algorithms:

```
Summation Formula:    1.2612e-14
DFT matrix:           6.9869e-15
fft:                  0
```

***Fig.26** Difference of the norms between the algorithms, N = 9*

From Fig.26 it is evident that the difference of the norms are very close to zero, therefore, all the algorithms generates the same result.

g)

Time it takes for each algorithm is calculated using *tic* and *toc* commands. The results are gathered in the Table 2.

Part	Time
Part (c)	0.000041 s
Part (d)	0.000074 s
Part (e)	0.000013 s

***Table 2:** Time it takes to compute different DFT approaches, N = 9*

Table 2 shows the time taken by three different DFT approaches for DFT of length N = 9. As expected, there is a significant variation in time across different methods. These times indicate that the MATLAB built-in FFT function is still the fastest, as expected, with the custom 9-pt FFT slightly slower than both the summation method and the built-in function. Despite the minimal time differences, the overall performance aligns with expectations: built-in FFTs are highly optimized, while custom methods like the 9-pt FFT have higher overheads. 9-pt FFT DIF algorithm is slower than the direct summation because array length is too small N = 9. In

Q1) *(k)* it is shown that for N = 32 FFT DIF algorithm is slightly slower than the direct summation but for N = 2^12 FFT DIF algorithm is significantly faster. Therefore, for a larger array like N = 3^8 the result could have been significantly faster.

Appendicies

Appendix A - Q1 whole code

```
close all
clear
clc

rng(2,"twister")
N = 32;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
steps_a_h(x);

rng(2,"twister")
N = 256;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
steps_a_h(x);

rng(2,"twister")
N = 2^12;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;
steps_a_h(x);

function steps_a_h(x)
N = length(x);
fprintf('N = %d \n', N);
n = 0:N-1;

figure;
subplot(2,1,1);
stem(n, real(x), 'filled');
title('Real Part of x[n]');
xlabel('n');
ylabel('Re\{x[n]\}');
xlim([0 N-1]);
grid on;

subplot(2,1,2);
stem(n, imag(x), 'filled');
title('Imaginary Part of x[n]');
xlabel('n');
ylabel('Im\{x[n]\}');
xlim([0 N-1]);
grid on;

t1 = tic;
Xsum = DFT_summation(x);
elapsed_time = toc(t1);
fprintf('Elapsed time for DFT_summation is %.6f seconds.\n', elapsed_time);
plotDFT(Xsum, 'Summation Formula')

t2 = tic;
X_mat = DFT_matrix(x);
```

```
elapsed_time = toc(t2);
fprintf('Elapsed time for DFT_matrix is %.6f seconds.\n', elapsed_time);
plotDFT(X_mat, 'DFT matrix')

t3 = tic;
X_DIT = FFT_DIT(x);
elapsed_time = toc(t3);
fprintf('Elapsed time for FFT_DIT is %.6f seconds.\n', elapsed_time);
plotDFT(X_DIT, 'FFT-DIT')

t4 = tic;
X_DIF = bitrevorder(FFT_DIF(x));
elapsed_time = toc(t4);
fprintf('Elapsed time for FFT_DIF is %.6f seconds.\n', elapsed_time);
plotDFT(X_DIF, 'FFT-DIF')

t5 = tic;
X_fft = fft(x);
elapsed_time = toc(t5);
fprintf('Elapsed time for fft is %.6f seconds.\n', elapsed_time);
plotDFT(X_fft, 'FFT')
fprintf('\n');

disp(['Summation Formula: ', num2str(norm(X_fft - Xsum))]);
disp(['DFT matrix: ', num2str(norm(X_fft - X_mat))]);
disp(['FFT-DIT: ', num2str(norm(X_fft - X_DIT))]);
disp(['FFT-DIF: ', num2str(norm(X_fft - X_DIF))]);
disp(['fft: ', num2str(norm(X_fft - X_fft))]);
fprintf('\n');
end

function plotDFT(X, str)
N = length(X);
n = 0:N-1;

figure;
subplot(2,1,1);
stem(n, abs(X), 'filled');
title(['Magnitude of X[k] using ', str]);
xlabel('k');
ylabel('|X[k]|');
xlim([0 N-1]);
grid on;

subplot(2,1,2);
stem(n, angle(X), 'filled');
title(['Phase of X[k] using ', str]);
xlabel('k');
ylabel('∠X[k] (radians)');
xlim([0 N-1]);
grid on;
end

function X = FFT_DIF(x)
N = length(x);
if N == 1
X = x;
else
X = x;
```

```
% Butterfly stage
for k = 1:N/2
temp = X(k);
X(k) = temp + X(k + N/2);
X(k + N/2) = (temp - X(k + N/2)) * exp(-1i * 2 * pi * (k - 1) / N);
end
% Recursive stage
X(1:N/2) = FFT_DIF(X(1:N/2));
X(N/2+1:N) = FFT_DIF(X(N/2+1:N));
end
end


function X = FFT_DIT(x)
N = length(x);
if N == 1
X = x;
else
% Divide
x_even = x(1:2:end);
x_odd = x(2:2:end);
% Conquer
X_even = FFT_DIT(x_even);
X_odd = FFT_DIT(x_odd);
% Combine
WN = exp(-1i * 2 * pi / N);
W = 1;
X = zeros(1, N);
for k = 1:N/2
X(k) = X_even(k) + W * X_odd(k);
X(k + N/2) = X_even(k) - W * X_odd(k);
W = W * WN;
end
end
end


function X = DFT_matrix(x)
N = length(x);
WN = exp(-1i * 2 * pi / N);
DFT_mat = zeros(N, N);
for k = 0:N-1
for n = 0:N-1
DFT_mat(k+1, n+1) = WN^(k * n);
end
end
X = (DFT_mat * x.').';
end


function X = DFT_summation(x)
N = length(x);
X = zeros(1, N);
for k = 0:N-1
for n = 0:N-1
X(k+1) = X(k+1) + x(n+1) * exp(-1i * 2 * pi * k * n / N);
end
end
end
```

Appendix B - Q2 whole code

```
close all
clear
clc

rng(2,"twister")
N = 9;
real_part = randn(1,N);
imag_part = randn(1,N);
x = real_part + 1i*imag_part;

fprintf('N = %d \n', N);
n = 0:N-1;

figure;
subplot(2,1,1);
stem(n, real(x), 'filled');
title('Real Part of x[n]');
xlabel('n');
ylabel('Re\{x[n]\}');
xlim([0 N-1]);
grid on;

subplot(2,1,2);
stem(n, imag(x), 'filled');
title('Imaginary Part of x[n]');
xlabel('n');
ylabel('Im\{x[n]\}');
xlim([0 N-1]);
grid on;

t1 = tic;
Xsum = DFT_summation(x);
elapsed_time = toc(t1);
fprintf('Elapsed time for DFT_summation is %.6f seconds.\n', elapsed_time);
plotDFT(Xsum, 'Summation Formula')

t2 = tic;
X_9pt_fft = nineptFFT(x);
elapsed_time = toc(t2);
fprintf('Elapsed time for 9-pt FFT is %.6f seconds.\n', elapsed_time);
plotDFT(X_9pt_fft, '9-pt FFT')

t3 = tic;
X_fft = fft(x);
elapsed_time = toc(t3);
fprintf('Elapsed time for fft is %.6f seconds.\n', elapsed_time);
plotDFT(X_fft, 'FFT')

fprintf('\n');
disp(['Summation Formula: ', num2str(norm(X_fft - Xsum))]);
disp(['DFT matrix: ', num2str(norm(X_fft - X_9pt_fft))]);
disp(['fft: ', num2str(norm(X_fft - X_fft))]);
fprintf('\n');

function X = nineptFFT(x)
N = length(x);
X = zeros(1, N);
```

```
for r = 0:(N/3 - 1)
for p = 0:2
temp = 0;
for n = 0:(N/3 - 1)
inner_sum = 0;
for l = 0:2
inner_sum = inner_sum + x(n + l*N/3 + 1) * exp(-1j*2*pi*p*l/3);
end
temp = temp + inner_sum * exp(-1j*2*pi*n*r/(N/3)) * exp(-1j*2*pi*p*n/N);
end
X(3*r + p + 1) = temp;
end
end
end


function X = DFT_summation(x)
N = length(x);
X = zeros(1, N);
for k = 0:N-1
for n = 0:N-1
X(k+1) = X(k+1) + x(n+1) * exp(-1i * 2 * pi * k * n / N);
end
end
end


function plotDFT(X, str)
N = length(X);
n = 0:N-1;

figure;
subplot(2,1,1);
stem(n, abs(X), 'filled');
title(['Magnitude of X[k] using ', str]);
xlabel('k');
ylabel('|X[k]|');
xlim([0 N-1]);
grid on;

subplot(2,1,2);
stem(n, angle(X), 'filled');
title(['Phase of X[k] using ', str]);
xlabel('k');
ylabel('∠X[k] (radians)');
xlim([0 N-1]);
grid on;
end
```