Bilkent University

Electrical and Electronics Department

EEE 424 Coding Assignment 1

31/03/2025

Fatih Mehmet Çetin - 22201689

Introduction:

This assignment mainly focuses on discrete fourier transform (DFT) and its' applications on signal processing through different methods. We use Matlab for this assignment. We implement different DFT methods and compare their efficiency.

This assignment consists of two parts. In the first part, we focus on the 2^n point DFT & FFT methods. In the second part, we develop an algorithm for 3^n point FFT.

"codass1driver.m" is the main matlab file for this assignment. It contains the plot information, tic-toc logic, and acts as the only driver file for the whole assignment. **(Appendix 1)**

"dft32.m" is the traditional DFT summation function for 32-point DFT. **(Appendix 2)**

"dft32matrixgenerator.m" is the function that generates the 32-pt DFT matrix. **(Appendix 3)**

"fft32dif.m" and **"fft32dit.m"** are the functions that contains the DIF FFT and DIT FFT algorithms for 32-pt DFT. **(Appendices 4 & 5)**

“dft256.m”, **“dft256matrixgenerator.m”**, **“fft256dif.m”** and **“fft256dit.m”** are basically the same functions mentioned above which are used for question 1 part 2. **(Appendices 6 to 9)**

“dft4096.m”, **“dft4096matrixgenerator.m”**, **“fft4096dif.m”** and **“fft4096dit.m”** are basically the same functions mentioned above which are used for question 1 part 3. **(Appendices 10 to 13)**

“dft9.m” is the function for traditional 9-point DFT using summation. **(Appendix 14)**

“fft9dif.m” is the function for 9-point FFT algorithm. **(Appendix 15)**

Assignment Work:

Question 1 Part 1:

Here you can see the real and imaginary parts of the random complex array we created whose length is 32 **(Figures 1.1 & 1.2)**.

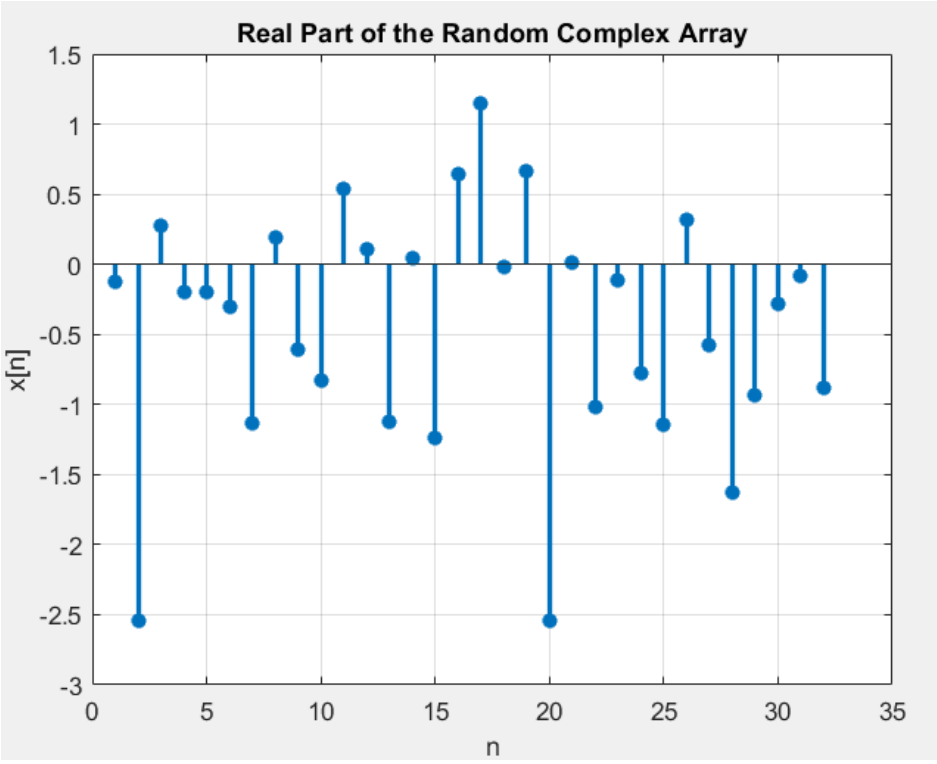

Figure 1.1: Real Part of the Random Array

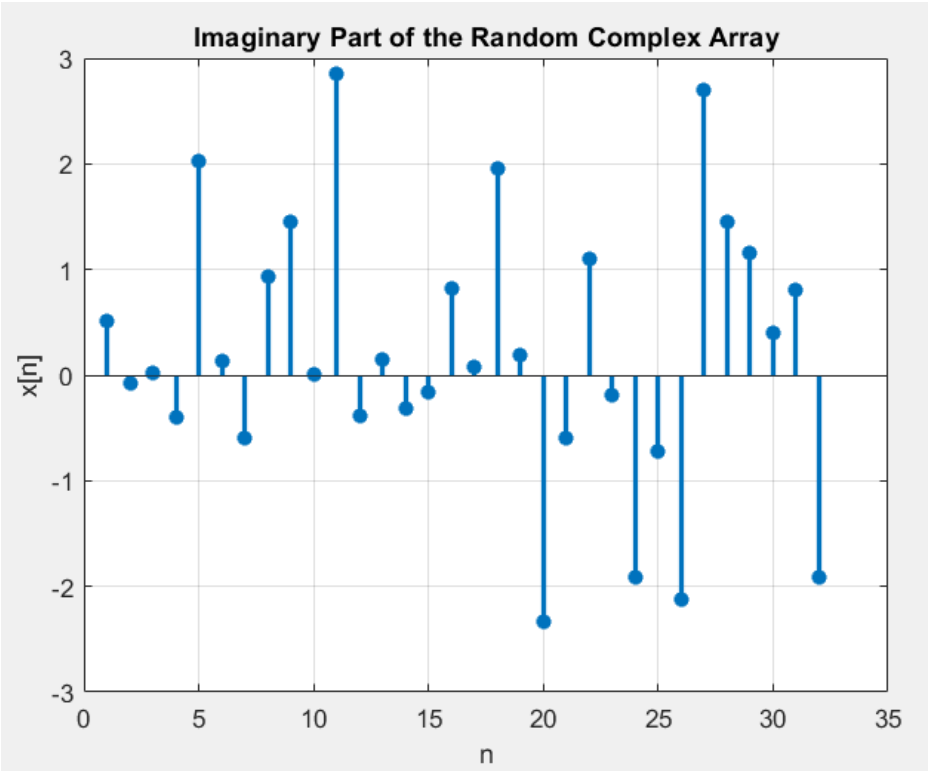

Figure 1.2: Imaginary Part of the Random Array

At this point, we calculated the DFT of the random complex array by 5 different methods. Table I gives some insight to how each method work.

TABLE I

Method 1	We compute the 32-pt DFT by directly using its definition in summation form
Method 2	We construct a DFT matrix and use it to compute the 32-pt DFT
Method 3	We implement FFT using decimation-in-time algorithm
Method 4	We implement FFT using decimation-in-frequency algorithm
Method 5	We use the built-in Matlab command for FFT

Here are the magnitude and phase plots of all the computed DFT's **(Figures 1.3 & 1.4)**.

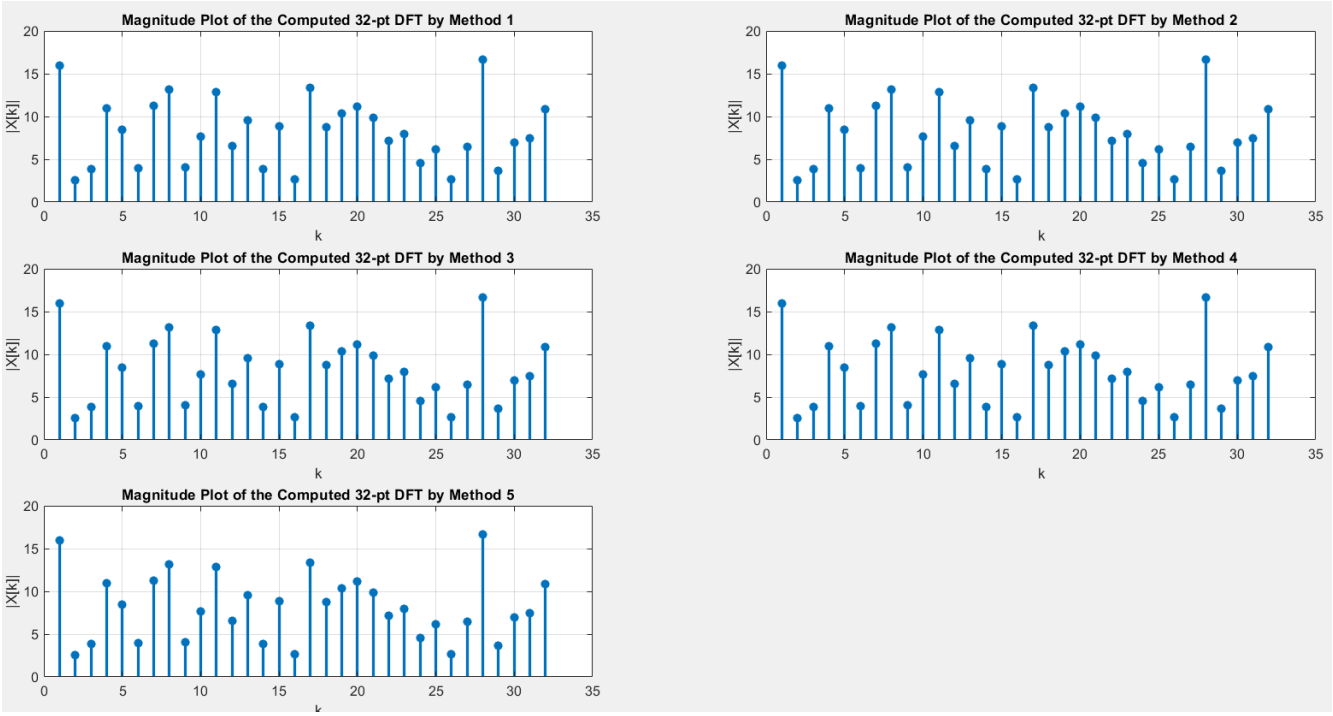

Figure 1.3: Magnitude Plot of the 32-Point DFT's

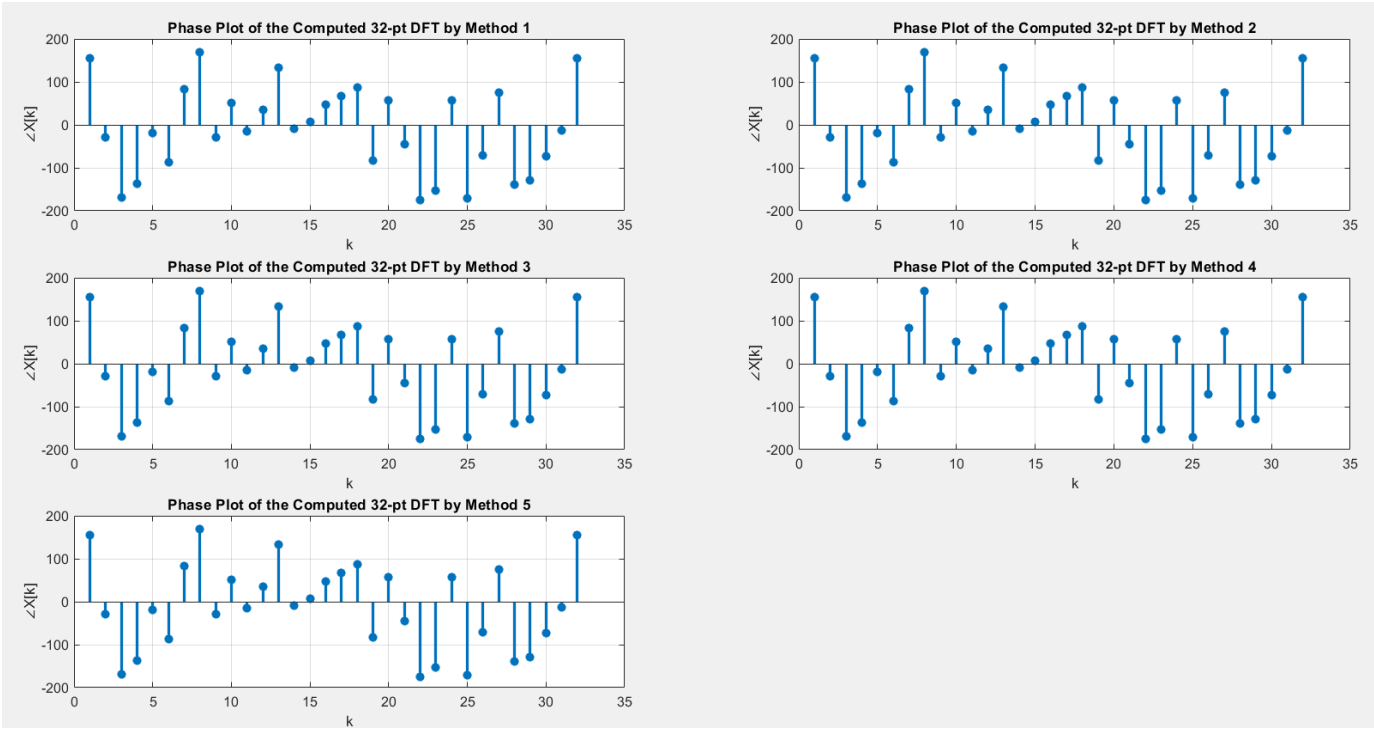

Figure 1.4: Phase Plot of the 32-Point DFT's

As expected, all the magnitude and phase graphs are equivalent to each other. Now, let's show that they are identical numerically as well **(Figure 1.5)**.

```
>> norms32 = [norm(a1-a2),norm(a1-a3),norm(a1-a4),norm(a1-a5)];
norms32 < 0.00001
%if the above line gives 4 logical 1's; then the algorithms are correct
%a1,a2,a3,a4,a5 are the 5 DFT's with 5 methods

ans =

  1×4 logical array

   1   1   1   1
```

Figure 1.5: Numerical Verification of the Equivalence of DFTs.

Question 1 Part 2:

Here you can see the real and imaginary parts of the random complex array we created whose length is 256 **(Figures 2.1 & 2.2)**.

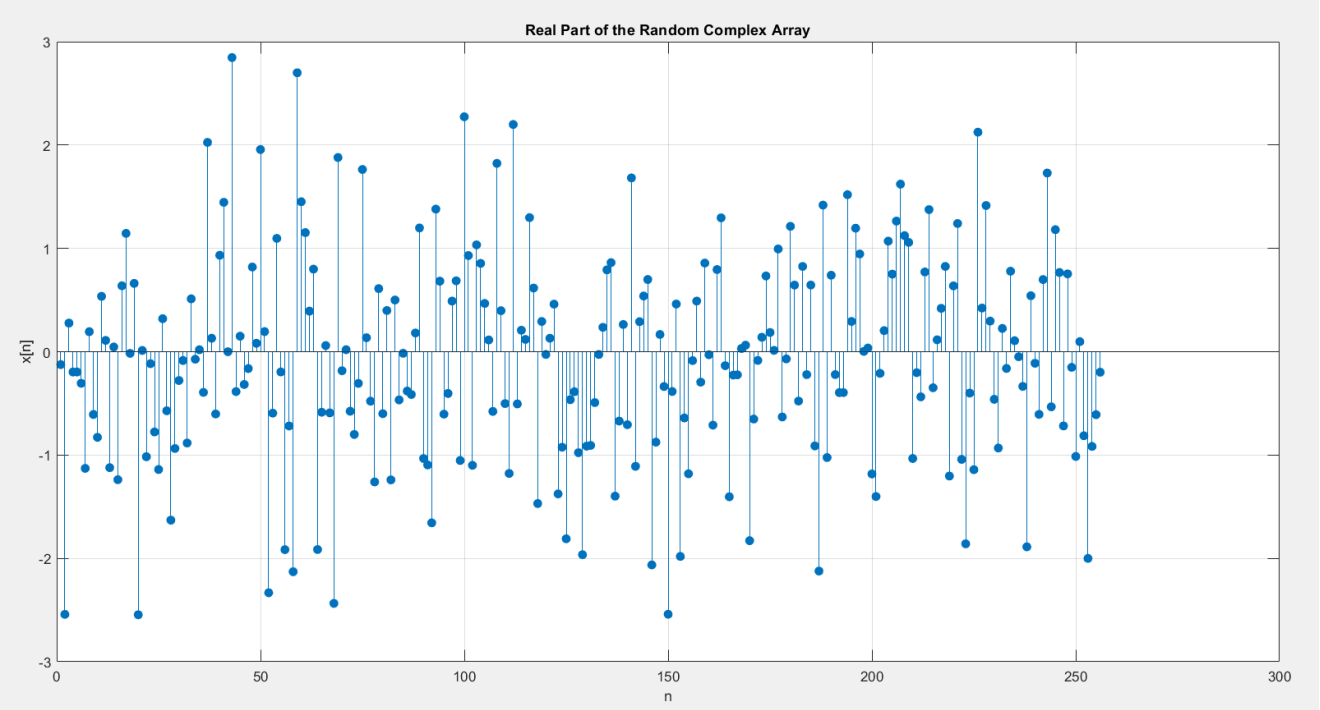

Figure 2.1: Real Part of the Random Array

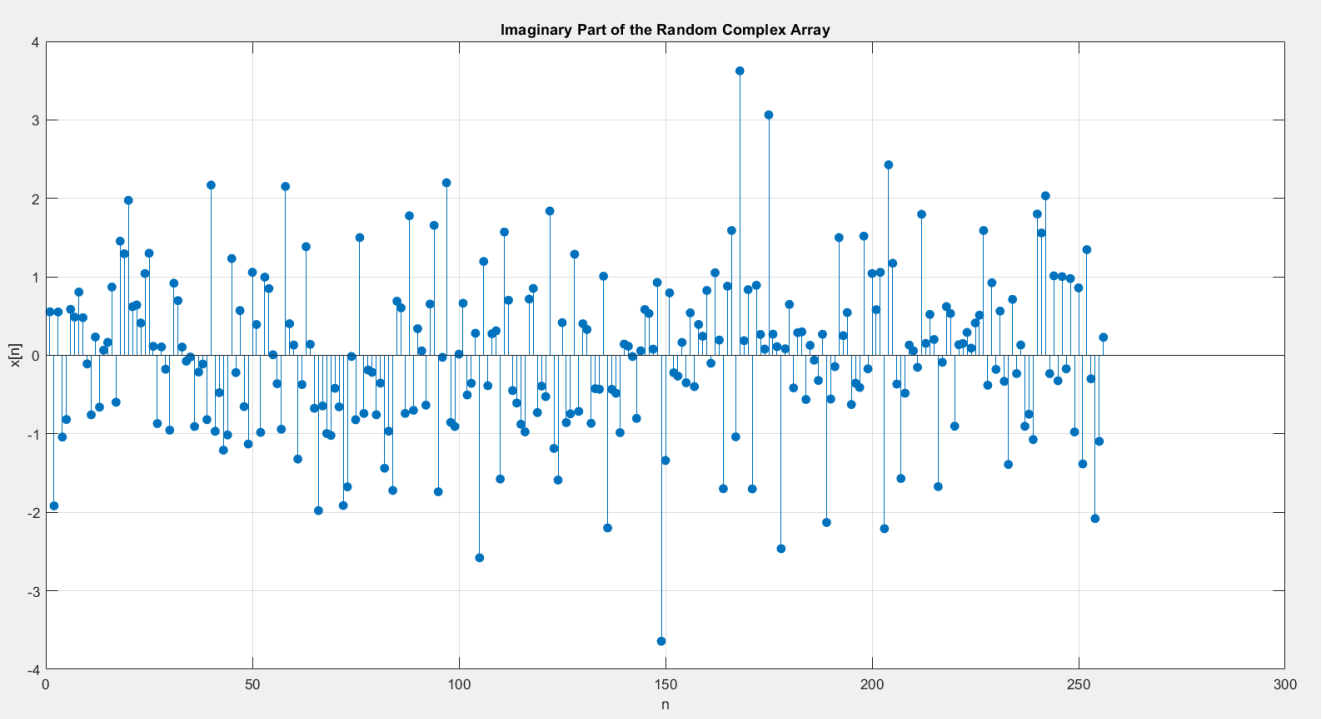

Figure 2.2: Imaginary Part of the Random Array

At this point, we calculated the DFT of the random complex array by 5 different methods. The methods are identical to the methods in the first part. Table I in the first part clearly explains each method.

Here are the magnitude and phase plots of all the computed DFT's **(Figures 2.3 & 2.4)**.

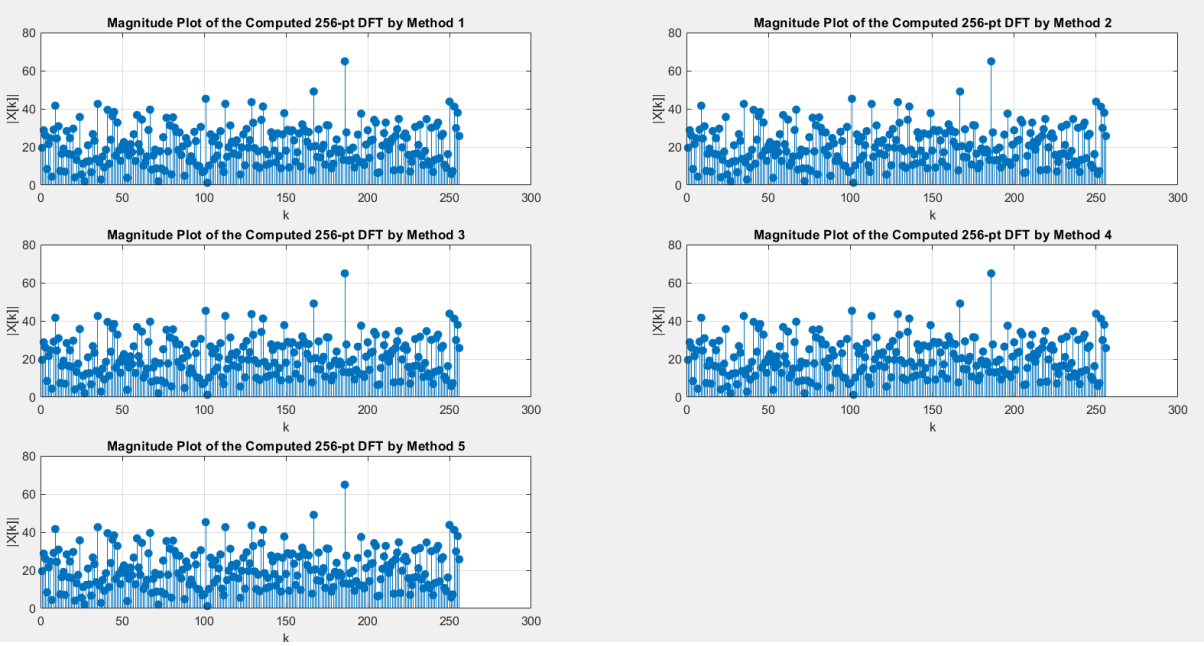

Figure 2.3: Magnitude Plot of the 256-Point DFT's

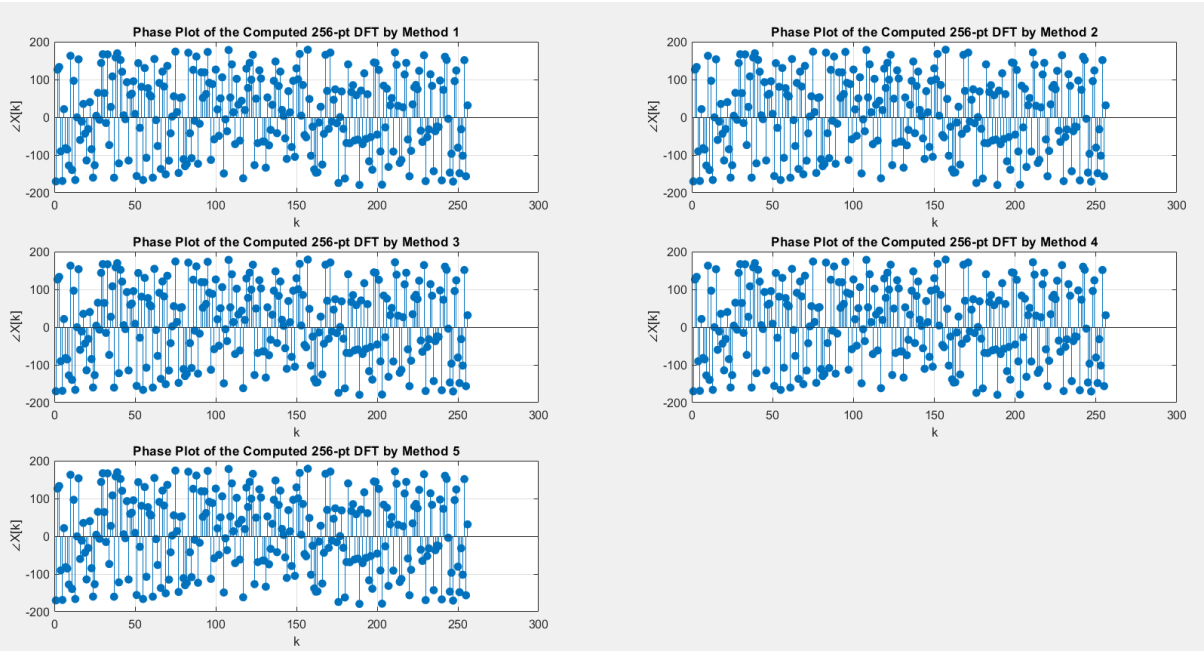

Figure 2.4: Phase Plot of the 256-Point DFT's

As expected, all the magnitude and phase graphs are equivalent to each other. Now, let's show that they are identical numerically as well **(Figure 2.5)**.

```
>> norms256 = [norm(b1-b2),norm(b1-b3),norm(b1-b4),norm(b1-b5)];
norms256 < 0.00001
%if the above line gives 4 logical 1's; then the algorithms are correct
%b1,b2,b3,b4,b5 are the 5 DFTs with 5 methods

ans =

  1×4 logical array

   1   1   1   1
```

Figure 2.5: Numerical Verification of the Equivalence of DFTs.

Question 1 Part 3:

Here you can see the real and imaginary parts of the random complex array we created whose length is 4096 **(Figures 3.1 & 3.2)**.

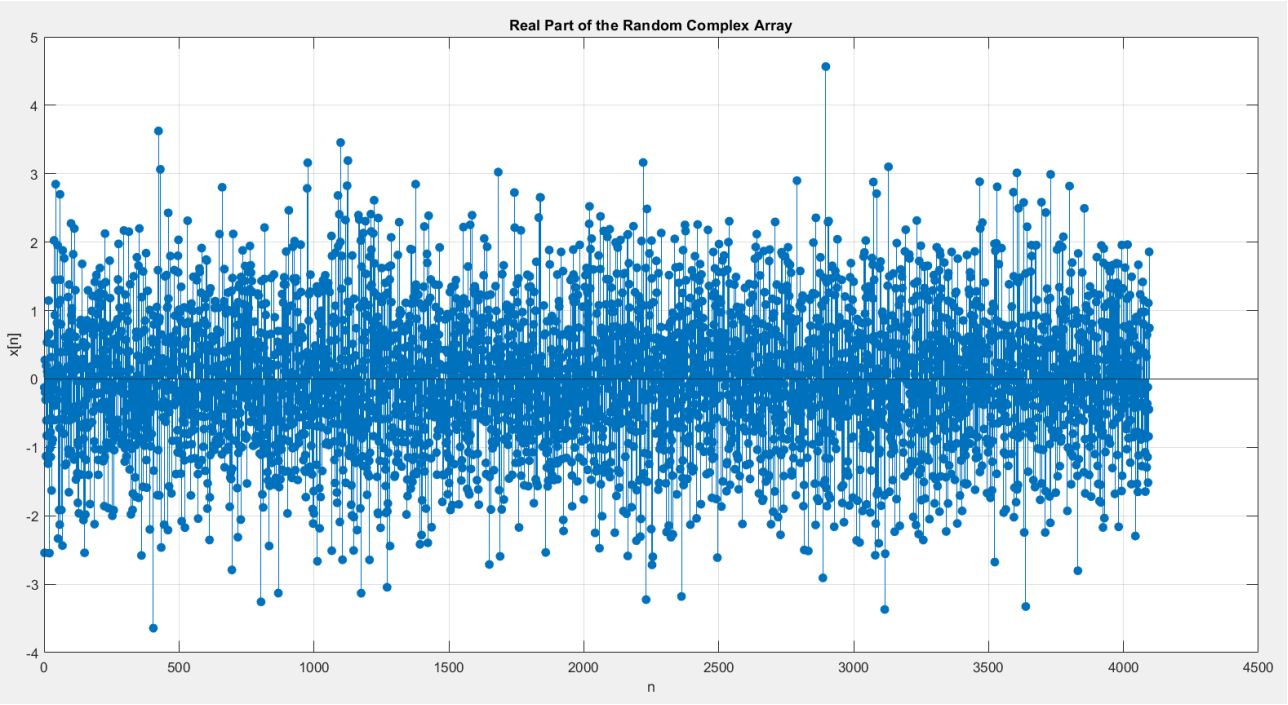

Figure 3.1: Real Part of the Random Array

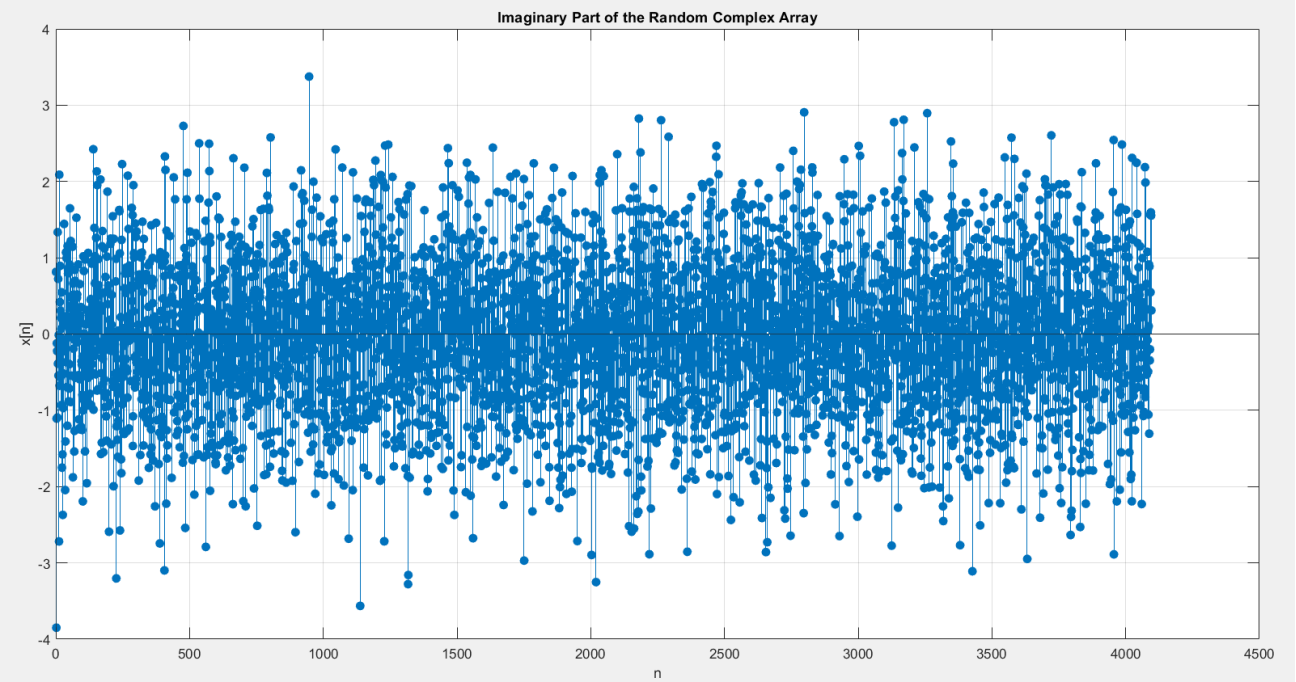

Figure 3.2: Imaginary Part of the Random Array

At this point, we calculated the DFT of the random complex array by 5 different methods. The methods are identical to the methods in the first part. Table I in the first part clearly explains each method.

Here are the magnitude and phase plots of all the computed DFT's **(Figures 3.3 & 3.4)**.

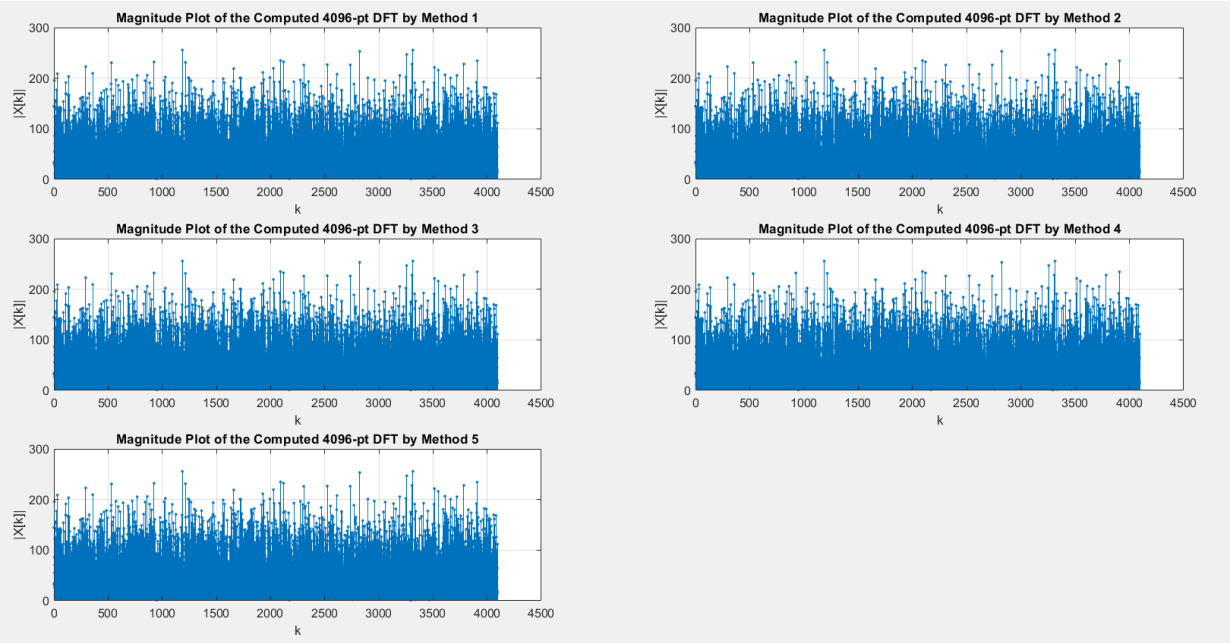

Figure 3.3: Magnitude Plot of the 4096-Point DFT's

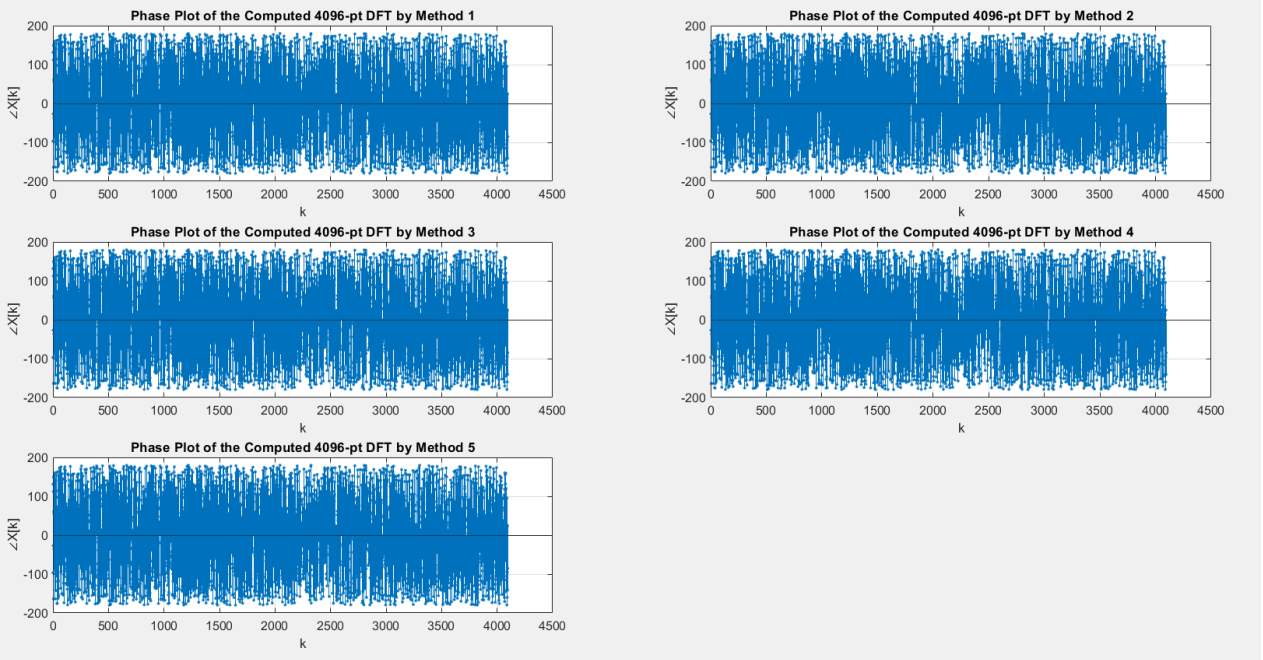

Figure 3.4: Phase Plot of the 4096-Point DFT's

As expected, all the magnitude and phase graphs are equivalent to each other. Now, let's show that they are identical numerically as well **(Figure 3.5)**.

```
>> norms4096 = [norm(c1-c2),norm(c1-c3),norm(c1-c4),norm(c1-c5)];
norms4096 < 0.00001
%if the above line gives 4 logical 1's; then the algorithms are correct
%c1,c2,c3,c4,c5 are the 5 DFTs with 5 methods

ans =

  1×4 logical array

   1   1   1   1
```

Figure 3.5: Numerical Verification of the Equivalence of DFTs.

Table II shows the time it takes in milliseconds for every single DFT operation with different methods and different length of input signals.

TABLE II

	N=32	**N=256**	**N=4096**
Method 1 (part b)	4.5 ms	8.6 ms	1898.7 ms

Method 2 (part d)	3.6 ms	9.7 ms	2093.1 ms
Method 3 (part e)	5.1 ms	2.5 ms	8.1 ms
Method 4 (part f)	2.1 ms	2.7 ms	7.8 ms
Method 5 (part g)	1.8 ms	1.9 ms	2.2 ms

We can make several observations from Table II. Firstly, as expected in every method the time to conduct the DFT operation increases as the length of the input signal increases. Secondly, FFT is definitely faster than traditional DFT operation. This is because the time complexity for traditional DFT is n^2 whereas it is $N*\log_2(N)$ for FFT. This increased efficiency shows itself much more powerfully with bigger input signals. Moving from traditional DFT to FFT have reduced the time of the operation from nearly 2 seconds to 2 milliseconds when the input's length was 4096. No wonder FFT is revolutionary even today. Also, we can observe that DIF FFT is slightly faster than DIT FFT. Generally speaking, using the DFT matrix can be slightly more efficient than directly summating DFT because of matrix optimization techniques. Nevertheless, in our case we counted the construction of the DFT matrix in the timer, therefore it took more time than traditional DFT. Our algorithms could not beat Matlab's FFT algorithm in terms of speed and efficiency. This is not a surprise because there is way more optimization techniques for my FFT code.

Question 2:

Here are the on-paper analytical solutions for the algorithm to compute a 3^n point FFT **(Figure 4.1)**. Table III shows the algorithm steps.

Table III

Step 1	Divide x[n] into three using indexes sorted by modulo 3
Step 2	Compute 3-Point DFTs seperately
Step 3	Apply W_9^{2k} and W_9^{k} coefficients
Step 4	Combine final results and re-order the array to get X[k]

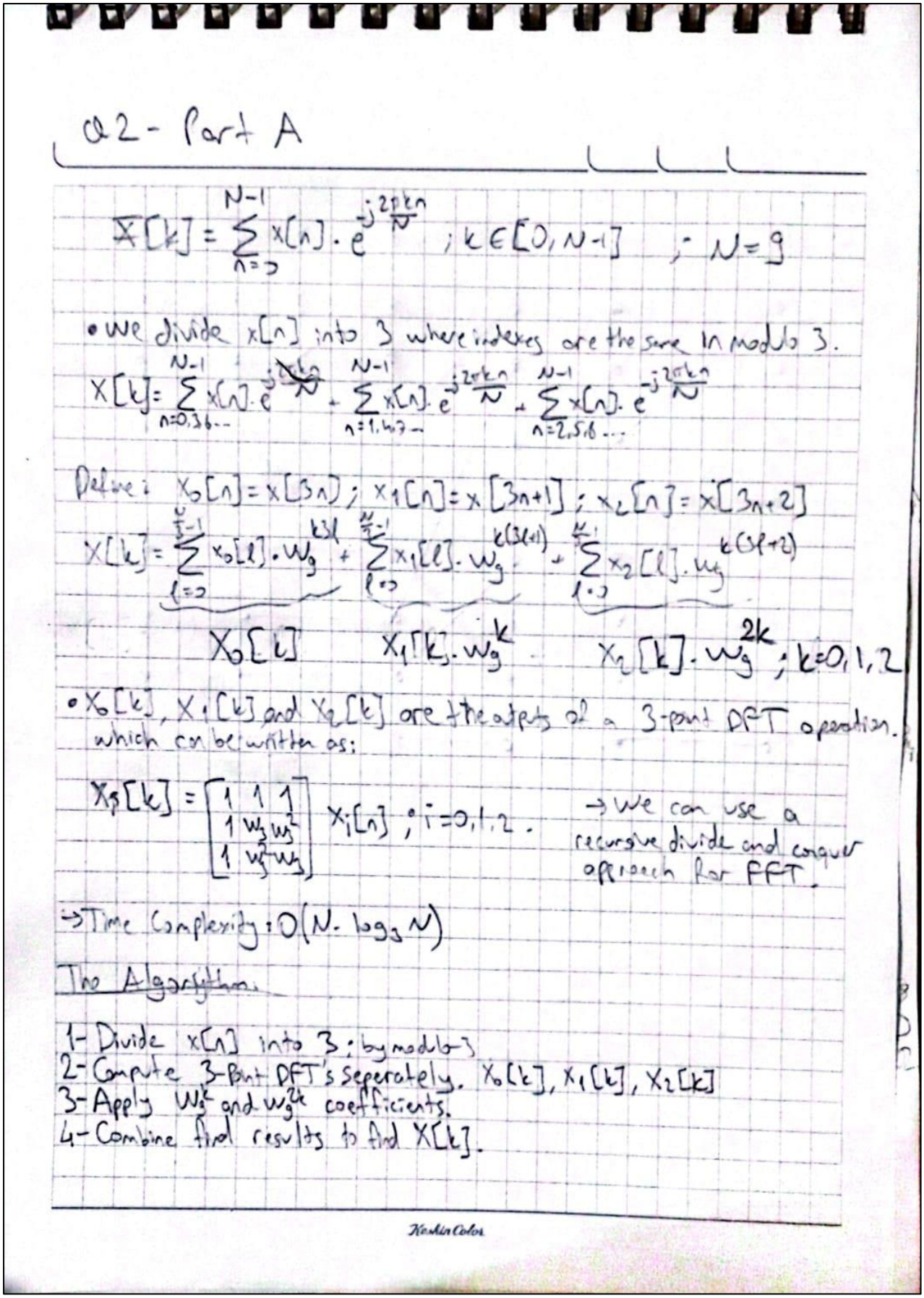

Figure 4.1: On-Paper Solutions to the Algorithm for 3^n point DFT

Here you can see the flow graph for the 9-point FFT algorithm **(Figure 4.2)**.

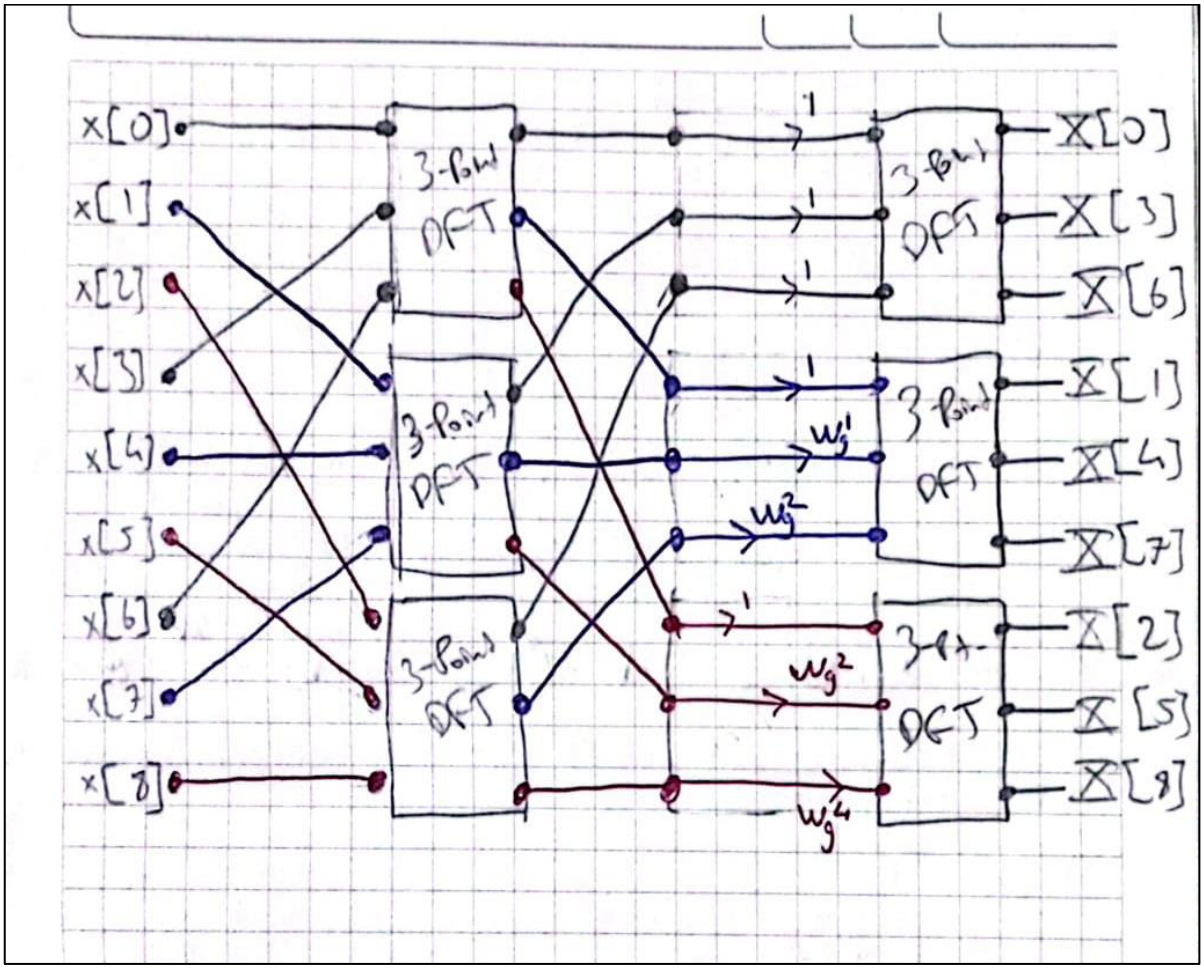

Figure 4.2: Flow Graph of 9-Point FFT

Now, we will compute the 9-Point DFT by three methods. Table IV explains each method in a detailed fashion.

Table IV

Method 1	We compute the 32-pt DFT by directly using its definition in summation form
Method 2	We implement the FFT algorithm we derived on-paper
Method 3	We use the built-in Matlab command for FFT

Here are the magnitude and phase plots of all the computed DFT's **(Figures 4.3 & 4.4)**.

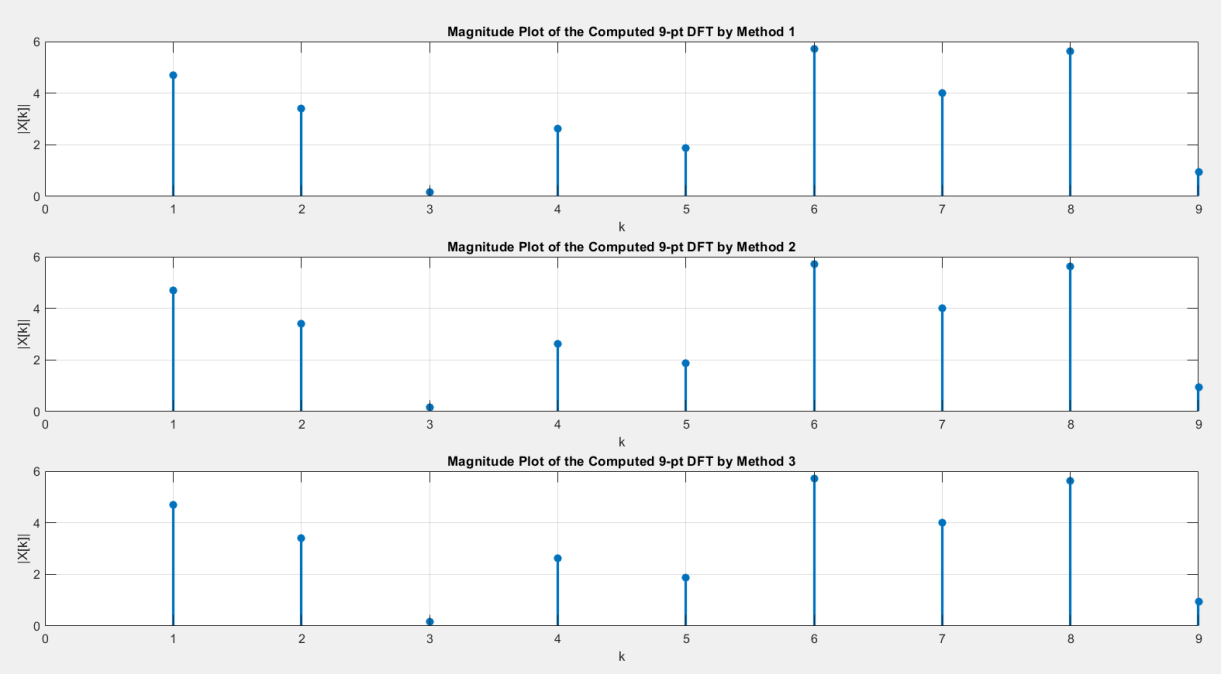

Figure 4.3: Magnitude Plot of the 9-Point DFT's

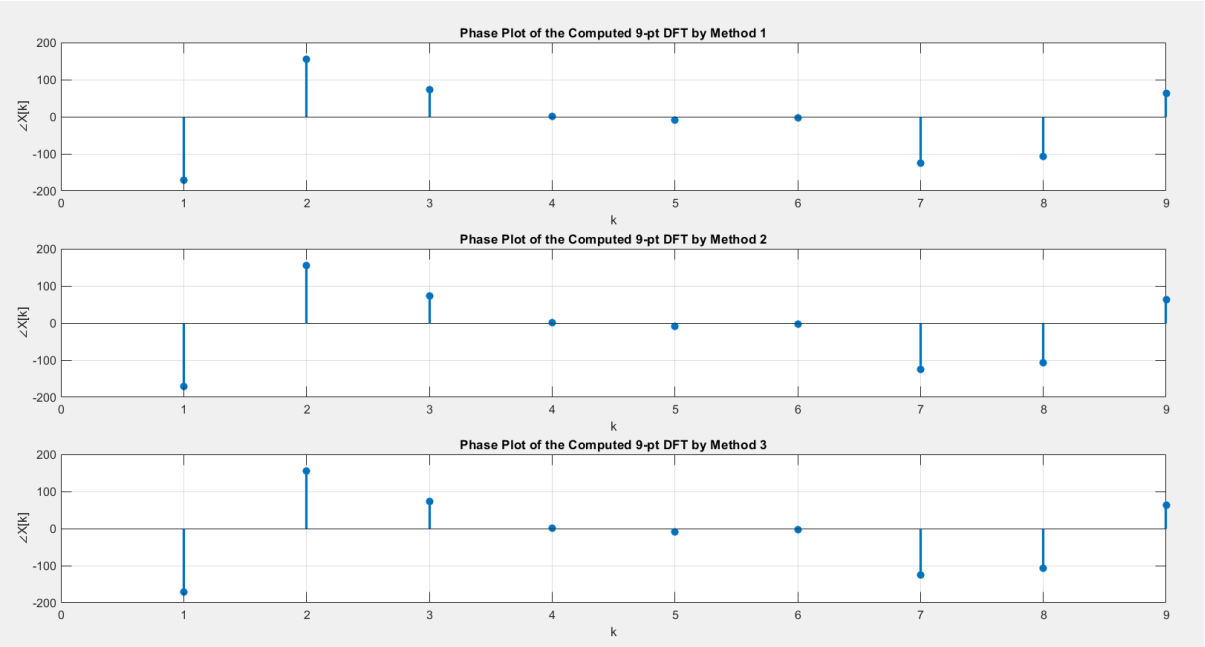

Figure 4.4: Phase Plot of the 9-Point DFT's

As expected, all the magnitude and phase graphs are equivalent to each other. Now, let's show that they are identical numerically as well **(Figure 4.5)**.

```
>> norms9 = [norm(d1-d2),norm(d1-d3)];
norms9 < 0.00001
%if the above line gives 2 logical 1's; then the algorithms are correct
%d1,d2,d3 are the 3 DFTs with 3 methods

ans =

  1×2 logical array

   1   1
```

Figure 3.5: Numerical Verification of the Equivalence of DFTs.

Table V shows the time it takes in milliseconds for every single 9-point DFT operation with different methods in question 2.

TABLE V

Method 1 (part c)	4.4 ms
Method 2 (part d)	9.2 ms
Method 3 (part e)	3.3 ms

We can make several observations from Table V. Firstly, Matlab's FFT algorithm definitely beats us and the traditional DFT. On the other hand, it is clear that our algorithm did not work as the way we intended it to work. Traditional DFT was way faster than our algorithm. This is due to the smallness of N. For bigger N, our FFT has the upperhand, but in small N traditional DFT might win against our algorithm.

Conclusion & Comments:

This assignment mainly focused on discrete fourier transform (DFT) and its' applications on signal processing through different methods. We used Matlab for this assignment. We implemented different DFT methods and compared their efficiency.

We definitely proved that FFT is significantly more efficient than traditional DFT in part 1. The time complexity reduction of FFT compared to DFT is phenomenal. Also, we observed that DIF FFT is slightly faster than DIT FFT in Matlab. In part 2 of the assignment, we observed that our 9-point FFT algorithm was way slower than traditional DFT. This was due to the smallness of the length of the DFT.

I believe the assignment is a total success. All the conditions were met. All the graphs were plotted. All the findings were found successfully. I think this lab was extremely helpful

in understanding non-standard FFT operations -3^n. I spent hours trying to come up with a working algorithm for part 2 and finally was able to successfully achieve my goal.

Appendices:

1. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/codass1driver.m
2. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft32.m
3. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft32matrixgenerator.m
4. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft32dif.m
5. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft32dit.m
6. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft256.m
7. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft256matrixgenerator.m
8. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft256dif.m
9. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft256dit.m
10. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft4096.m
11. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft4096matrixgenerator.m
12. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft4096dif.m
13. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft4096dit.m
14. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/dft9.m
15. https://github.com/fmcetin7/Bilkent-EEE-424/blob/main/Coding%20Assignment%201/fft9dif.m