# LAB 2

## Sliding DFT

| Name | ID |
|---|---|
| Jolie Atef Abdallah | 20010441 |
| Mary Ayman Fawzy | 20011139 |
| Nardin Refaat | 20012076 |
| Veronica Melad Sadek | 20011090 |
| Asmaa Hassan Mokhtar | 19015430 |

## Introduction

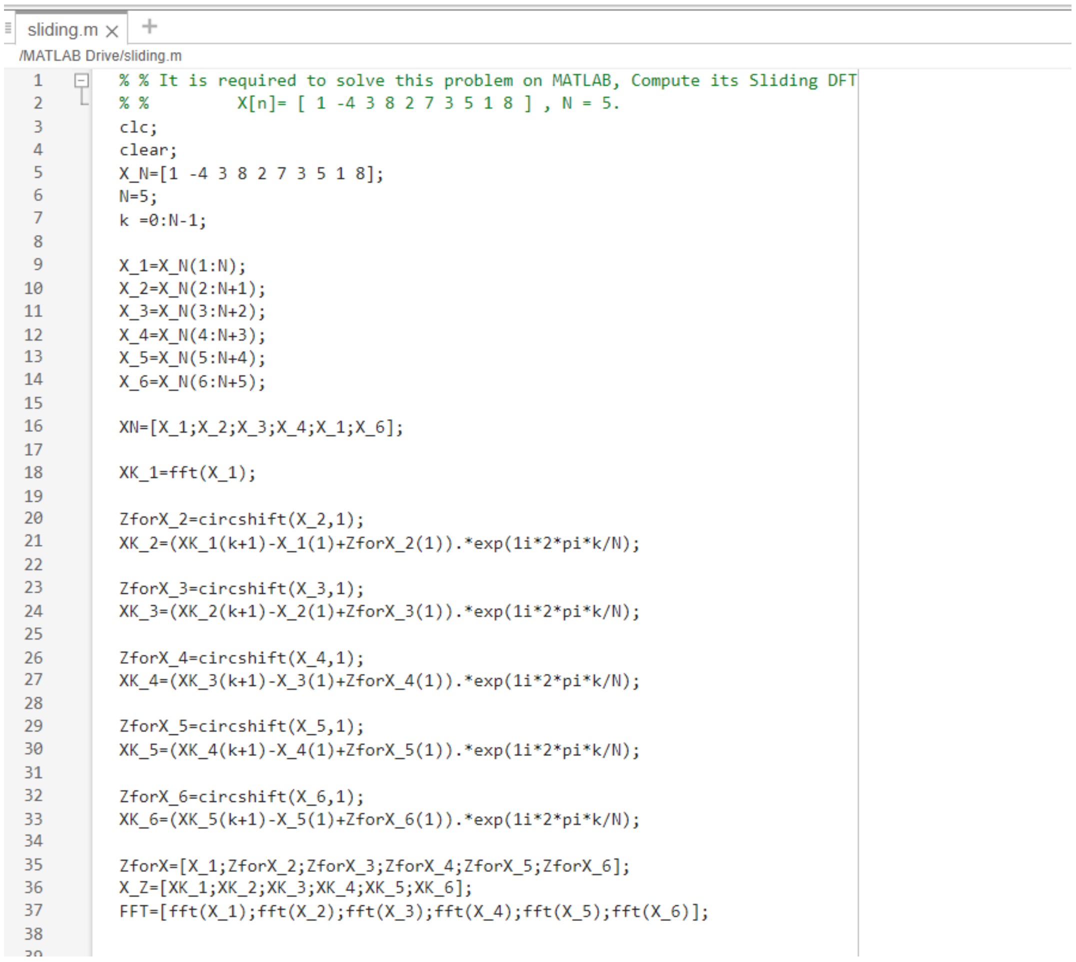
The Sliding Discrete Fourier Transform (SDFT) is a signal processing technique that allows for the real-time computation of the Discrete Fourier Transform (DFT) as a signal evolves over time. It is particularly useful in applications where there is a need for continuous monitoring and analysis of a changing signal, such as in audio processing, communication systems, and various other time-series data analysis tasks.

The traditional DFT computes the frequency components of an entire signal or a fixed-length window. In contrast, the Sliding DFT continuously updates the frequency representation of the signal as it moves through time. This is achieved by applying the DFT to overlapping windows of the signal, incrementally shifting the window through the signal samples.

## Requirements

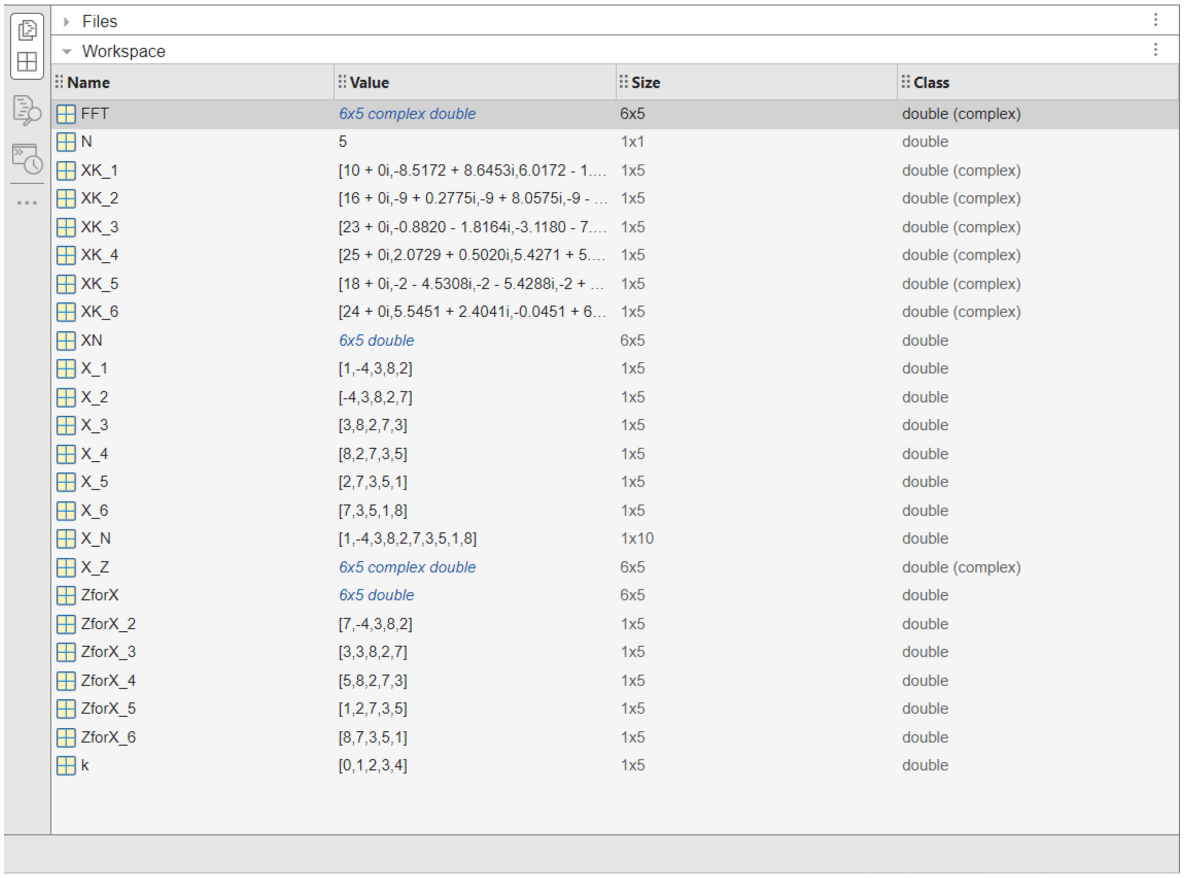
It is required to solve this problem on MATLAB, Compute its Sliding DFT
X[n]= [ 1 -4 3 8 2 7 3 5 1 8]

## Code



```
% % It is required to solve this problem on MATLAB, Compute its Sliding DFT
% %         X[n]= [ 1 -4 3 8 2 7 3 5 1 8 ] , N = 5.
clc;
clear;
X_N=[1 -4 3 8 2 7 3 5 1 8];
N=5;
k =0:N-1;

X_1=X_N(1:N);
X_2=X_N(2:N+1);
X_3=X_N(3:N+2);
X_4=X_N(4:N+3);
X_5=X_N(5:N+4);
X_6=X_N(6:N+5);

XN=[X_1;X_2;X_3;X_4;X_1;X_6];

XK_1=fft(X_1);

ZforX_2=circshift(X_2,1);
XK_2=(XK_1(k+1)-X_1(1)+ZforX_2(1)).*exp(1i*2*pi*k/N);

ZforX_3=circshift(X_3,1);
XK_3=(XK_2(k+1)-X_2(1)+ZforX_3(1)).*exp(1i*2*pi*k/N);

ZforX_4=circshift(X_4,1);
XK_4=(XK_3(k+1)-X_3(1)+ZforX_4(1)).*exp(1i*2*pi*k/N);

ZforX_5=circshift(X_5,1);
XK_5=(XK_4(k+1)-X_4(1)+ZforX_5(1)).*exp(1i*2*pi*k/N);

ZforX_6=circshift(X_6,1);
XK_6=(XK_5(k+1)-X_5(1)+ZforX_6(1)).*exp(1i*2*pi*k/N);

ZforX=[X_1;ZforX_2;ZforX_3;ZforX_4;ZforX_5;ZforX_6];
X_Z=[XK_1;XK_2;XK_3;XK_4;XK_5;XK_6];
FFT=[fft(X_1);fft(X_2);fft(X_3);fft(X_4);fft(X_5);fft(X_6)];
```

## Workspace

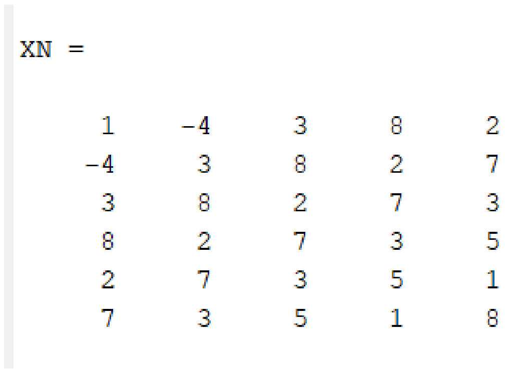


| Name | Value | Size | Class |
|---|---|---|---|
| FFT | 6x5 complex double | 6x5 | double (complex) |
| N | 5 | 1x1 | double |
| XK_1 | [10 + 0i,-8.5172 + 8.6453i,6.0172 - 1.… | 1x5 | double (complex) |
| XK_2 | [16 + 0i,-9 + 0.2775i,-9 + 8.0575i,-9 - … | 1x5 | double (complex) |
| XK_3 | [23 + 0i,-0.8820 - 1.8164i,-3.1180 - 7.… | 1x5 | double (complex) |
| XK_4 | [25 + 0i,2.0729 + 0.5020i,5.4271 + 5.… | 1x5 | double (complex) |
| XK_5 | [18 + 0i,-2 - 4.5308i,-2 - 5.4288i,-2 + … | 1x5 | double (complex) |
| XK_6 | [24 + 0i,5.5451 + 2.4041i,-0.0451 + 6… | 1x5 | double (complex) |
| XN | 6x5 double | 6x5 | double |
| X_1 | [1,-4,3,8,2] | 1x5 | double |
| X_2 | [-4,3,8,2,7] | 1x5 | double |
| X_3 | [3,8,2,7,3] | 1x5 | double |
| X_4 | [8,2,7,3,5] | 1x5 | double |
| X_5 | [2,7,3,5,1] | 1x5 | double |
| X_6 | [7,3,5,1,8] | 1x5 | double |
| X_N | [1,-4,3,8,2,7,3,5,1,8] | 1x10 | double |
| X_Z | 6x5 complex double | 6x5 | double (complex) |
| ZforX | 6x5 double | 6x5 | double |
| ZforX_2 | [7,-4,3,8,2] | 1x5 | double |
| ZforX_3 | [3,3,8,2,7] | 1x5 | double |
| ZforX_4 | [5,8,2,7,3] | 1x5 | double |
| ZforX_5 | [1,2,7,3,5] | 1x5 | double |
| ZforX_6 | [8,7,3,5,1] | 1x5 | double |
| k | [0,1,2,3,4] | 1x5 | double |

## Method

### Step 1:

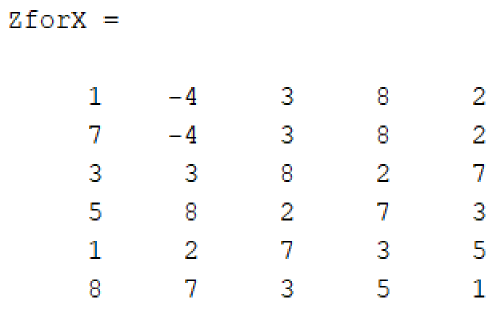
Make a window in the signal with Length(N=5) and slide along the signal



```
XN =

     1    -4     3     8     2
    -4     3     8     2     7
     3     8     2     7     3
     8     2     7     3     5
     2     7     3     5     1
     7     3     5     1     8
```

### Step 2:

The first segment is the same while performing a circular shift to the other segments.



```
ZforX =

     1    -4     3     8     2
     7    -4     3     8     2
     3     3     8     2     7
     5     8     2     7     3
     1     2     7     3     5
     8     7     3     5     1
```

### Note

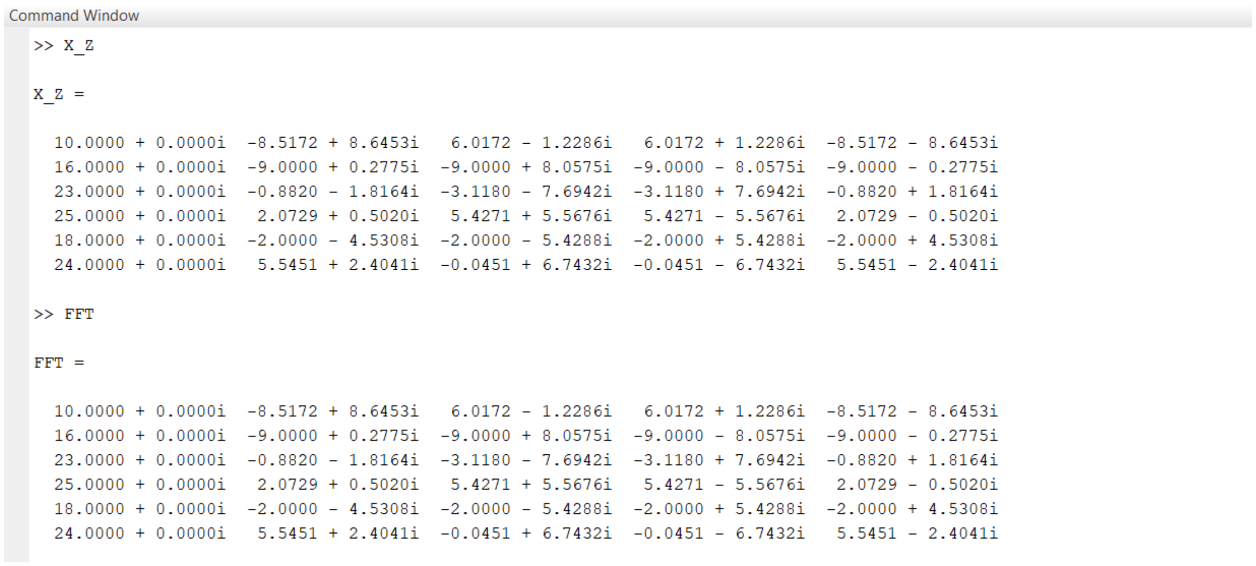
We notice that after circular convolution the segment last 4 samples of x(n) is the same as the last 4 samples of x(n-1) before convolution.

**Step 3**

Applying the following equation to all samples

XK(n)=(XK(n-1) (k+1)-X_(n-1) (1) +circonv(X(n)) (1)). *Exp(j*2*pi*k/N).

This is recursive equation to compute the N-point DFT at time index n using the DFT at the previous time index n-1 and the input samples at the current time index n and the past sample.



Command Window

```
>> X_Z

X_Z =

   10.0000 + 0.0000i  -8.5172 + 8.6453i   6.0172 - 1.2286i   6.0172 + 1.2286i  -8.5172 - 8.6453i
   16.0000 + 0.0000i  -9.0000 + 0.2775i  -9.0000 + 8.0575i  -9.0000 - 8.0575i  -9.0000 - 0.2775i
   23.0000 + 0.0000i  -0.8820 - 1.8164i  -3.1180 - 7.6942i  -3.1180 + 7.6942i  -0.8820 + 1.8164i
   25.0000 + 0.0000i   2.0729 + 0.5020i   5.4271 + 5.5676i   5.4271 - 5.5676i   2.0729 - 0.5020i
   18.0000 + 0.0000i  -2.0000 - 4.5308i  -2.0000 - 5.4288i  -2.0000 + 5.4288i  -2.0000 + 4.5308i
   24.0000 + 0.0000i   5.5451 + 2.4041i  -0.0451 + 6.7432i  -0.0451 - 6.7432i   5.5451 - 2.4041i

>> FFT

FFT =

   10.0000 + 0.0000i  -8.5172 + 8.6453i   6.0172 - 1.2286i   6.0172 + 1.2286i  -8.5172 - 8.6453i
   16.0000 + 0.0000i  -9.0000 + 0.2775i  -9.0000 + 8.0575i  -9.0000 - 8.0575i  -9.0000 - 0.2775i
   23.0000 + 0.0000i  -0.8820 - 1.8164i  -3.1180 - 7.6942i  -3.1180 + 7.6942i  -0.8820 + 1.8164i
   25.0000 + 0.0000i   2.0729 + 0.5020i   5.4271 + 5.5676i   5.4271 - 5.5676i   2.0729 - 0.5020i
   18.0000 + 0.0000i  -2.0000 - 4.5308i  -2.0000 - 5.4288i  -2.0000 + 5.4288i  -2.0000 + 4.5308i
   24.0000 + 0.0000i   5.5451 + 2.4041i  -0.0451 + 6.7432i  -0.0451 - 6.7432i   5.5451 - 2.4041i
```

**Note**

we can notice that after applying this equation to the sampled signal it gets the same results when we use DFT transform.

## Advantages:

1.Real-time Processing:

•Sliding DFT is particularly useful in real-time applications where data is continuously arriving, and you need to update the frequency content of the signal on the fly.

•It avoids the need to process an entire signal at once, making it suitable for streaming data or situations where processing time is critical.

2.Efficient Memory Usage:

•Instead of computing the DFT of the entire signal, the sliding DFT updates the frequency components as new data points arrive. This can be more memory-efficient, especially for large datasets.

3.Adaptability to Changing Signals:

•In applications where the signal characteristics change over time, the sliding DFT allows for adaptability by continuously updating the frequency components.

4.Reduced Computational Complexity:

•The sliding DFT can be more computationally efficient compared to recomputing the entire DFT when only a small portion of the signal changes.

5.Frequency Tracking:

•Sliding DFT is useful for tracking changes in the frequency content of a signal over time, making it suitable for applications such as audio processing, vibration analysis, and communication systems.

## Usages:

1.Audio Processing:

•Sliding DFT is applied in real-time audio processing for tasks such as pitch detection, musical instrument recognition, and audio compression.

2.Communication Systems:

•In communication systems, the sliding DFT can be used for adaptive filtering, spectrum analysis, and tracking changes in the frequency components of a signal.

3.Vibration Analysis:

•For monitoring and analyzing vibrations in mechanical systems, the sliding DFT can be used to track changes in the frequency content, helping to detect faults or anomalies.

4.Radar and Sonar Systems:

•In radar and sonar applications, the sliding DFT can be employed for real-time analysis of signal reflections, allowing for adaptive processing and tracking of moving targets.

5.Power Systems Monitoring:

•Sliding DFT can be used to monitor the frequency content of power signals in electrical systems, helping to detect and analyze disturbances or faults.