DSP –Lab Manual-3 – Discrete Fourier transform

Table of Contents

There are built-in functions available for computation of DFT and IDFT. These functions make use of FFT algorithms, which are computationally highly efficient compared to direct computation of DFT and IDFT.

Functions described below are part of the MATLAB Signal Processing Toolbox. The FFT function in Matlab is an algorithm published in 1965 by J.W.Cooley and J.W.Tuckey for efficiently calculating the DFT. [Ref.1]

[Ref.1] James W. Cooley and John W. Tukey, An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of Computation Vol. 19, No. 90, pp. 297-301, April 1965.

fft - Discrete Fourier transform

fft(X) is the discrete Fourier transform (DFT) of vector X.

fft(X,N) is the N-point fft, padded with zeros if X has less than N points and truncated if it has more.

In MATLAB,

- For length N input vector x, the DFT is a length N vector X, with elements

```
                N
X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
                n=1
```

- The inverse DFT (computed by IFFT) is given by

```
                   N
x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
                   k=1
```

ifft -Inverse discrete Fourier transform

ifft(X) is the inverse discrete Fourier transform of X.

ifft(X,N) is the N-point inverse transform.

```
x = [1 2 3 4];

y1 = fft(x)
```

```
y1 = 1×4 complex
  10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 + 0.0000i  -2.0000 - 2.0000i
```

```
y2 = fft(x,8)
```

```
y2 = 1×8 complex
  10.0000 + 0.0000i  -0.4142 - 7.2426i  -2.0000 + 2.0000i   2.4142 - 1.2426i ⋯
```

```
z1 = ifft (y1)
```

```
z1 = 1×4
     1     2     3     4
```

```
z2 = ifft (y2)
```

```
z2 = 1×8
    1.0000    2.0000    3.0000    4.0000    0.0000   -0.0000         0   -0.0000
```

dftmtx - Discrete Fourier transform matrix

In addition to above functions, the function dftmtx(N) can be used to compute N-by-N DFT matrix.

*dftmtx(N) is the N-by-N complex matrix of values around the unit-circle whose inner product with a column vector of length N yields the discrete Fourier transform of the vector. If X is a column vector of length N, then dftmtx(N)*X yields the same result as FFT(X); however, FFT(X) is more efficient.*

The inverse discrete Fourier transform matrix is CONJ(dftmtx(N))/N.

An interesting example is D = dftmtx(4)

```
D = dftmtx(4)
```

```
D = 4×4 complex
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 - 1.0000i  -1.0000 + 0.0000i   0.0000 + 1.0000i
   1.0000 + 0.0000i  -1.0000 + 0.0000i   1.0000 + 0.0000i  -1.0000 + 0.0000i
   1.0000 + 0.0000i   0.0000 + 1.0000i  -1.0000 + 0.0000i   0.0000 - 1.0000i
```

which returns D =   [1   1   1   1

1 -i -1   i

1 -1   1 -1

1   i -1 -i]

Let us calcualte the DFT by using Equation of DFT -

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

For illustration, we will take the example input sequence having length =4. Hence,

$$X[k] = \sum_{n=0}^{3} x[n] W_4^{kn}$$

And

$$X[k] = [x(0) \quad x(1) \quad x(2) \quad x(3)] \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix}$$

Then,

```
% Program to compute DFT of the sequence
x = [1 2 3 4];
N=4;
n = [0:1:N-1];k = [0:1:N-1];
WN = exp(-j*2*pi/N);
nk = n'*k;
WNnk = WN.^nk;
Xk =  x*WNnk
```

```
Xk = 1×4 complex
  10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 - 2.0000i
```

```
% Program to compute IDFT of the sequence
Xk =   [10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 - 2.0000i];
N=4;
n = [0:1:N-1];k = [0:1:N-1];
WN = exp(-j*2*pi/N);
nk = n'*k;
WNnk = WN.^-nk;
xn =  (Xk*WNnk)/N
```

```
xn = 1×4 complex
   1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 - 0.0000i   4.0000 - 0.0000i
```

Same result can also be obtained by using dftmtx(N) as follows:

```
Xk_mat = x *dftmtx(4)
```

```
Xk_mat = 1×4 complex
```

```
   10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 + 0.0000i  -2.0000 - 2.0000i
```

```
xn_mat = Xk_mat*conj(dftmtx(N))/N
```

```
xn_mat = 1×4
     1     2     3     4
```

Convolution

conv - Convolution and polynomial multiplication.

C = conv(A, B) convolves vectors A and B. The resulting vector is length MAX([LENGTH(A)+LENGTH(B)-1,LENGTH(A),LENGTH(B)]). If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

```
x = [2 1 2 1];
h = [1 2 3 4];
y = conv(x,h)
```

```
y = 1×7
     2     5    10    16    12    11     4
```

cconv Modulo-N circular convolution -

C = cconv(A, B, N) circularly convolves vectors A and B. The resulting vector is length N. If omitted, N defaults to LENGTH(A)+LENGTH(B)-1.

When N = LENGTH(A)+LENGTH(B)-1, the circular convolution is equivalent to the linear convolution computed by CONV.

```
x = [2 1 2 1];
h = [1 2 3 4];
y = cconv(x,h,4)
```

```
y = 1×4
    14    16    14    16
```

Linear convolution using circular convolution:-

Let x[n] be the sequence of length l1, and h[n] be the sequence of length l2, then the linear convolution of the two is,

$$y[n] = x[n] * h[n] \qquad \text{length } N = l1 + l2 - 1$$

For the linear convolution using DFT, define two new sequences,

$$x_e[n] = \begin{cases} x[n] & 0 \le n \le l1 - 1 \\ 0 & l1 \le n \le N - 1 \end{cases}$$

and

$$h_e[n] = \begin{cases} h[n] & 0 \le n \le l2 - 1 \\ 0 & l2 \le n \le N - 1 \end{cases}$$

Then, the result of circular convolving the sequences $x_e[n]$ and $h_e[n]$ produces same result as that of linear convolution. For example, the linear convolution of the vectors x and h of the previous example can be obtained as follows:

```
y_lc = cconv(x,h,7)
```

```
y_lc = 1×7
    2.0000    5.0000   10.0000   16.0000   12.0000   11.0000    4.0000
```

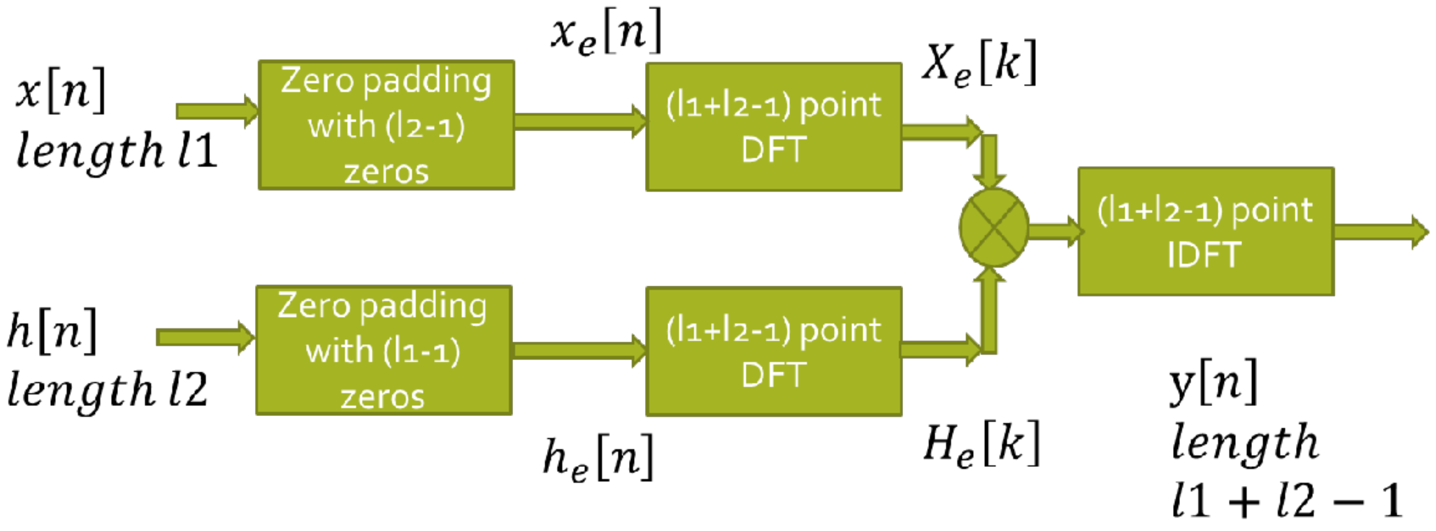
Or simply using

```
y_lc = cconv(x,h)
```

```
y_lc = 1×7
    2.0000    5.0000   10.0000   16.0000   12.0000   11.0000    4.0000
```

```
% %If you omit n, it defaults to length(a)+length(b)-1.
% When n = length(a)+length(b)-1, the circular convolution is equivalent to the
% linear convolution computed with conv.
```

The result of circular convolution and linear convolution can be obtained using DFT as well. For achieving linear convolution of the two sequences, use the steps provided in the Figure below.



```
Xk = fft(x,7);
Hk = fft(h,7);
Product_XH = Xk.*Hk;
Lin_conv = ifft(Product_XH,7)
```

```
Lin_conv = 1×7
    2.0000    5.0000   10.0000   16.0000   12.0000   11.0000    4.0000
```

For achiving circulation convolution

```
Xk = fft(x);
Hk = fft(h);
```

```
Product_XH = Xk.*Hk;
circular_conv = ifft(Product_XH)
```

```
circular_conv = 1×4
    14    16    14    16
```

Plotting Spectrum using DFT/FFT

Equation of DFT is given as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{kn}{N}}$$

This array can be interpreted as follows: -

- X[0] represents DC frequency component
- Next N/2 terms are positive frequency components with X[N/2] being the Nyquist frequency (which is equal to half of sampling frequency (also known as folding frequency)
- Next N/2 -1 are negative frequency components (note: negative frequency components are the phasors rotating in opposite direction, they can be optionally omitted depending on the application).

FFT result in MATLAB can be interpreted using figure as follows:

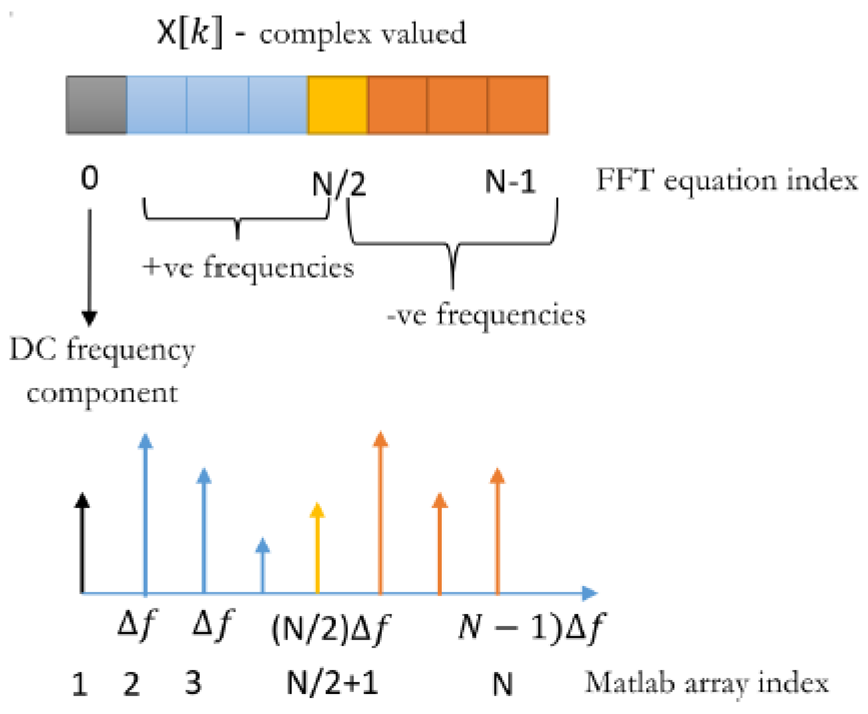
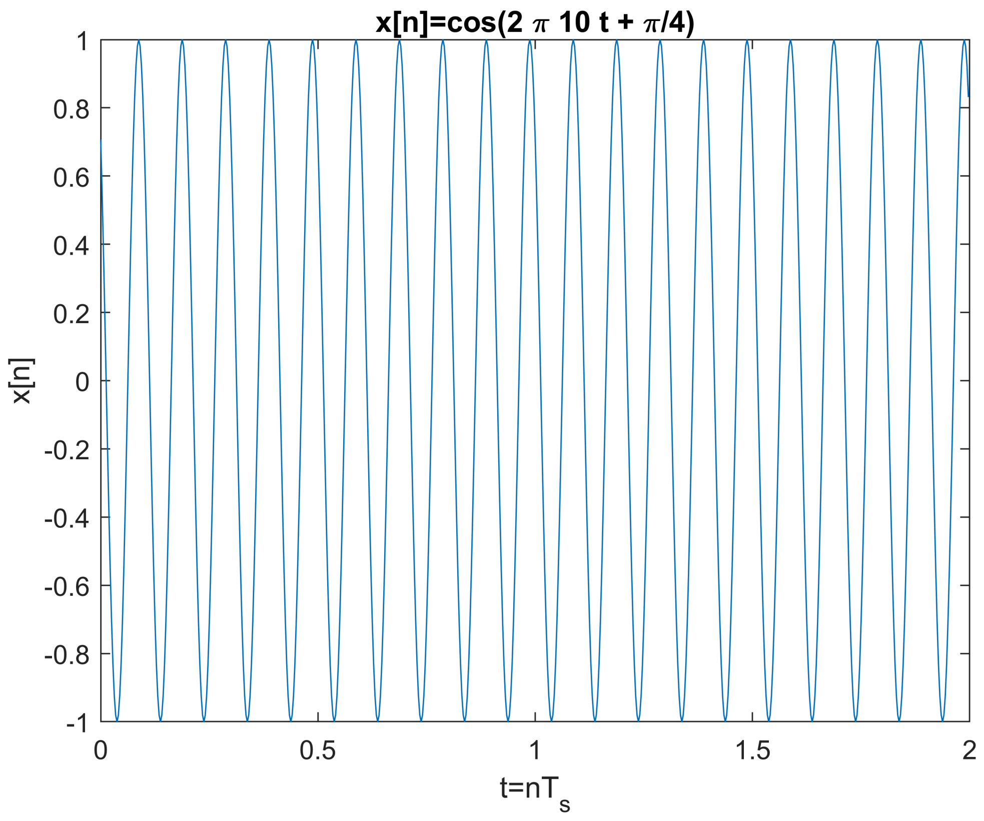


Figure-1 - Interpretation of frequencies

Let’s assume that the x[n] is the time domain cosine signal of frequency fc = 10 Hz that is sampled at a frequency fs = 32*fc.

```
fc=10;%frequency of the carrier
```

```
fs=32*fc;%sampling frequency with oversampling factor=32
t=0:1/fs:2-1/fs;%2 seconds duration
x=cos(2*pi*fc*t + pi/4);%time domain signal (real number)
plot(t,x); %plot the signal
title('x[n]=cos(2 \pi 10 t + \pi/4)'); xlabel('t=nT_s'); ylabel('x[n]');
```



Tanking the N point DFT (N=256)

```
N=256; %FFT size
X = fft(x,N);
% N-point complex DFT, output contains DC at index 1
% Nyquist frequency at N/2+1 th index
% positive frequencies from index 2 to N/2
% negative frequencies from index N/2+1 to N
```

Note: The FFT length should be sufficient to cover the entire length of the input signal. If N is less than the length of the input signal, the input signal will be truncated when computing the FFT. In our case, the cosine wave is of 2 seconds duration and it will have 640 points (a 10Hz frequency wave sampled at 32 times oversampling factor will have 2*32*10 = 640 samples in 2 seconds of the record). Since our input signal is periodic, we can safely use N = 256 point FFT.

Due to Matlab's index starting at 1, the DC component of the FFT decomposition is present at index 1.

```
X(1)  % 3.9053e-15 (approximately equal to zero)
```

```
ans = 3.9053e-15
```

Note that the index for the raw FFT are integers from 1 --> N. We need to process it to convert these integers to frequencies. That is where the sampling frequency counts. Each point or bin in the FFT output array is spaced by the frequency resolution Δf , that is calculated as,

$$\Delta f = \frac{\text{fs}}{N}$$

where, fs is the sampling frequency and N is the FFT size that is considered. Thus, for our example, each point in the array is spaced by the frequency resolution

$$\Delta f = \frac{\text{fs}}{N} = \frac{320}{256} = 1.25 \text{ Hz}$$

The 10Hz cosine signal will register a spike at the 8th sample (10/1.25=8) - located at index 9.

```
abs(X(8:10))  %display samples 7 to 9  ans = 0.0000 128.0000 0.0000
```

```
ans = 1×3
    0.0000  128.0000    0.0000
```

Therefore, from the frequency resolution, the entire frequency axis can be computed as

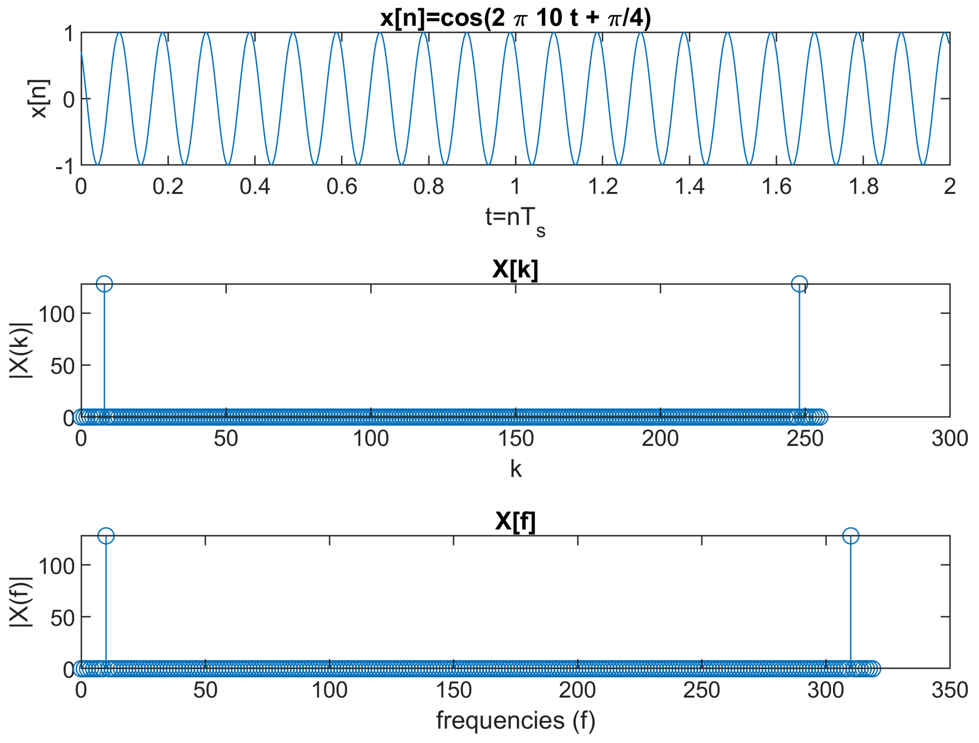
```
%calculate frequency bins with FFT
df=fs/N %frequency resolution
```

```
df = 1.2500
```

```
sampleIndex = 0:N-1; %raw index for FFT plot
f=sampleIndex*df; %x-axis index converted to frequencies
```

Now, plot the absolute value of the FFT against frequencies - the resulting plot is shown below. (Note that scaling factor of 1/N can be used.)

```
subplot(3,1,1);plot(t,x);hold on; %plot the signal
title('x[n]=cos(2 \pi 10 t + \pi/4)'); xlabel('t=nT_s'); ylabel('x[n]');
subplot(3,1,2); stem(sampleIndex,abs(X)); %sample values on x-axis
title('X[k]'); xlabel('k'); ylabel('|X(k)|');
subplot(3,1,3); stem(f,abs(X)); %x-axis represent frequencies
title('X[f]'); xlabel('frequencies (f)'); ylabel('|X(f)|');
```



In the above figure, we see that the frequency axis starts with DC, followed by positive frequency terms which is in turn followed by the negative frequency terms. To introduce proper order in the x-axis, one can use FFTshift function Matlab, which arranges the frequencies in order: negative frequencies --> DC --> positive frequencies.

For even N, the original order returned by FFT is as follows (note: here, all indices corresponds to Matlab's index)

- X[1] represents DC frequency component
- X[2] to X[N/2] terms are positive frequency components
- X[N/2+1] to X[N] terms are considered as negative frequency components

FFTshift shifts the DC component to the center of the spectrum. This process is illustrated as follows:

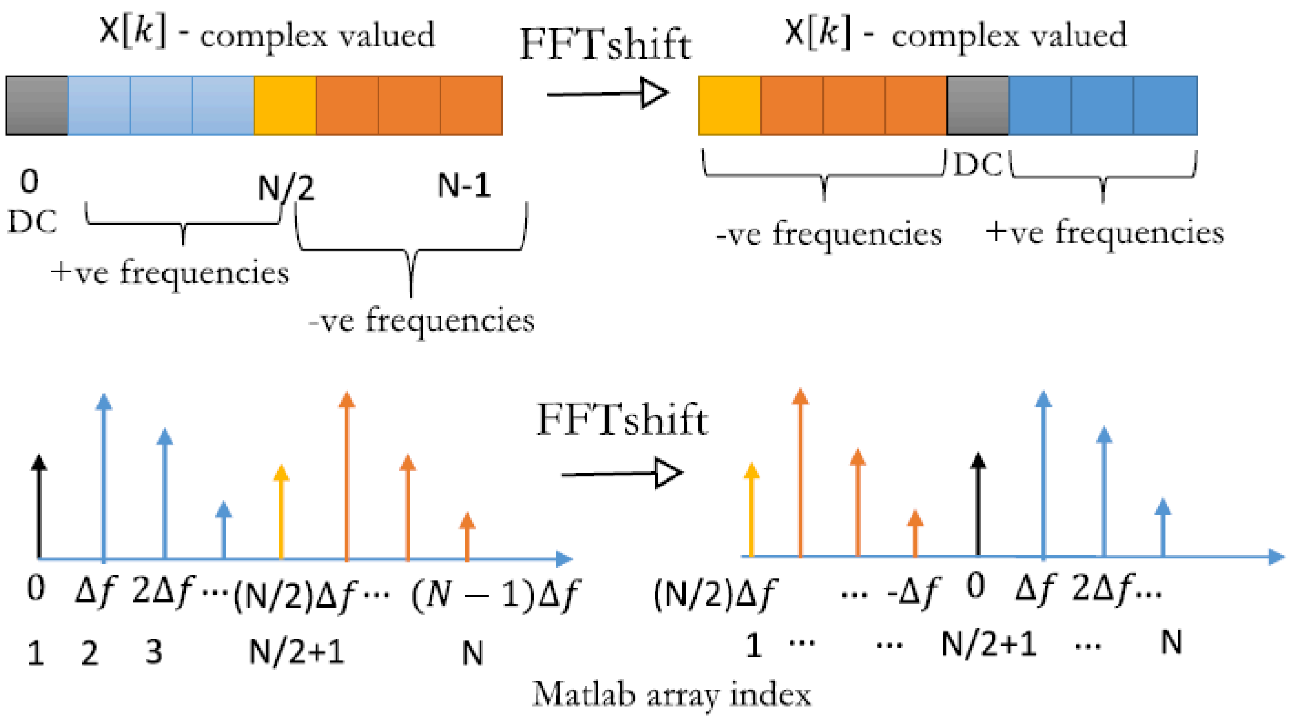


Figure 2 - Use of fftshift function

Therefore, when N is even, ordered frequency axis is set as

$$f = \Delta f\left[-\frac{N}{2} : 1 : \frac{N}{2} - 1\right] = \frac{fs}{N}\left[-\frac{N}{2} : 1 : \frac{N}{2} - 1\right]$$

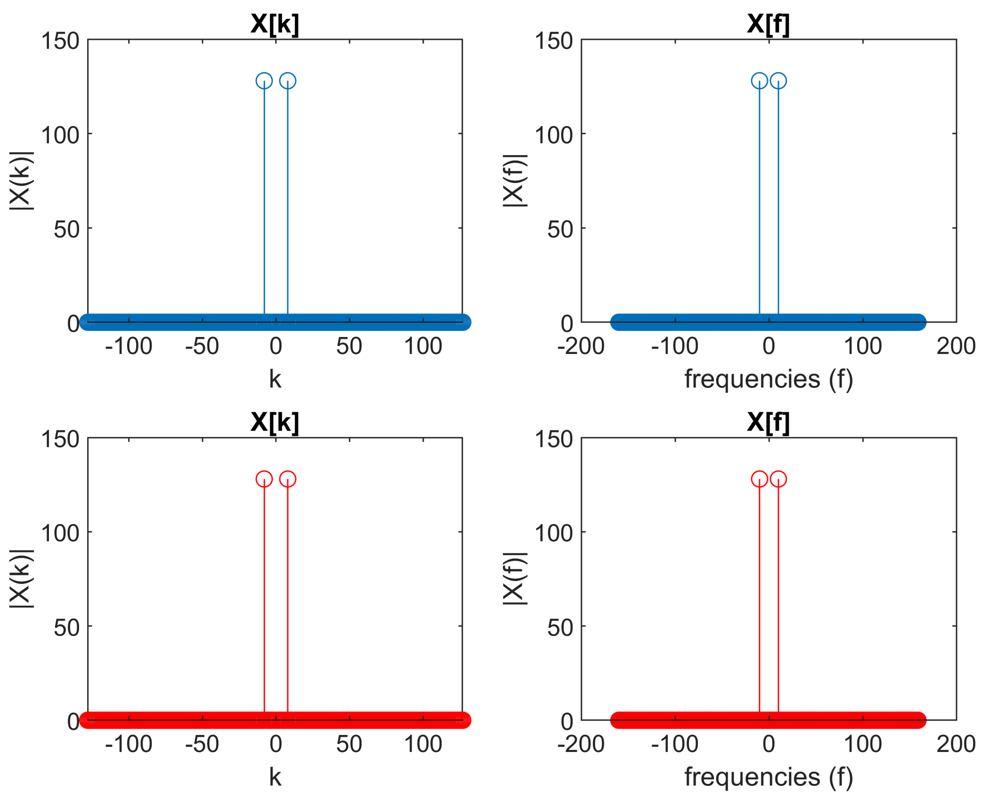
When N is odd, the ordered frequency axis should be set as

$$f = \Delta f\left[-\frac{N+1}{2} : 1 : \frac{N+1}{2} - 1\right] = \frac{fs}{N}\left[-\frac{N+1}{2} : 1 : \frac{N+1}{2} - 1\right]$$

Following code snippet performs the shifting using both the manual method and using the Matlab’s in-built command (fftshift).

```
X1 = [(X(N/2+1:N)) X(1:N/2)]; %order frequencies without using fftShift
df=fs/N; %frequency resolution
sampleIndex = -N/2:N/2-1; %raw index for FFT plot
f=sampleIndex*df; %x-axis index converted to frequencies
figure;
subplot(2,2,1);stem(sampleIndex,abs(X1));
title('X[k]'); xlabel('k'); ylabel('|X(k)|');
subplot(2,2,2);stem(f,abs(X1));
title('X[f]'); xlabel('frequencies (f)'); ylabel('|X(f)|');%frequencies on x-axis

X2 = fftshift(X);%order frequencies by using fftshift
subplot(2,2,3); stem(sampleIndex,abs(X2),'r') %sample index on x-axis
title('X[k]'); xlabel('k'); ylabel('|X(k)|');
subplot(2,2,4);stem(f,abs(X2),'r')
title('X[f]'); xlabel('frequencies (f)'); ylabel('|X(f)|');%frequencies on x-axis
```



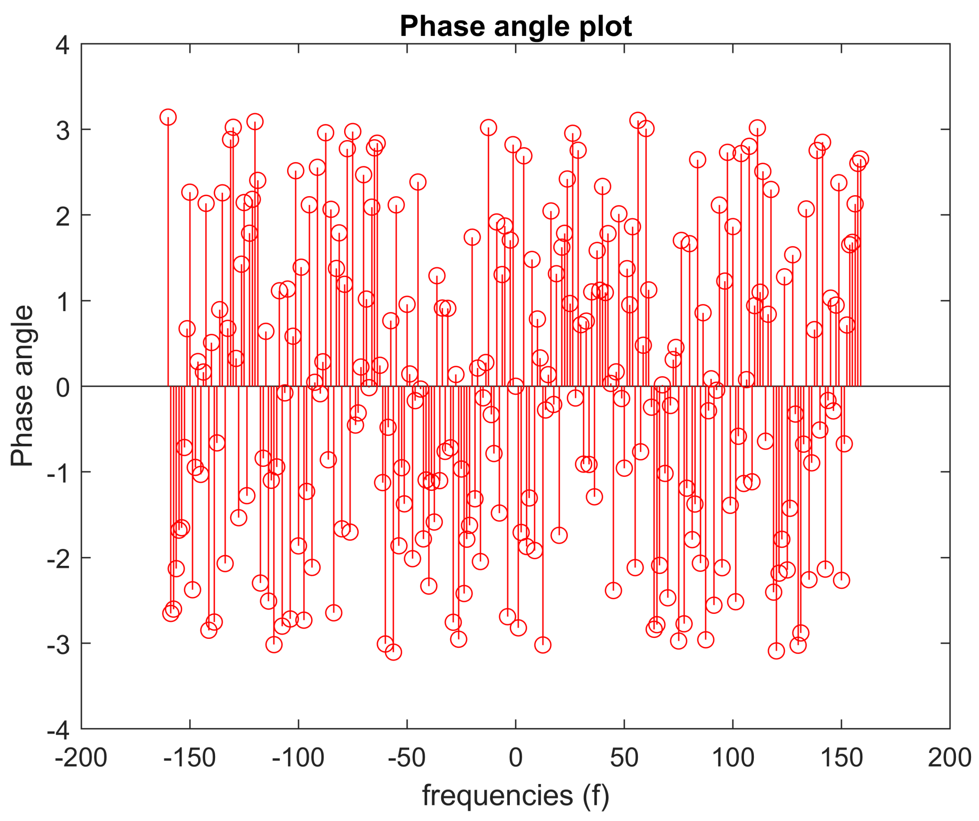
Phase spectrum

The phase of the spectral components are computed as

$$\angle X[k] = \tan^{-1}\left(\frac{X_{\mathrm{im}}}{X_{\mathrm{ra}}}\right)$$

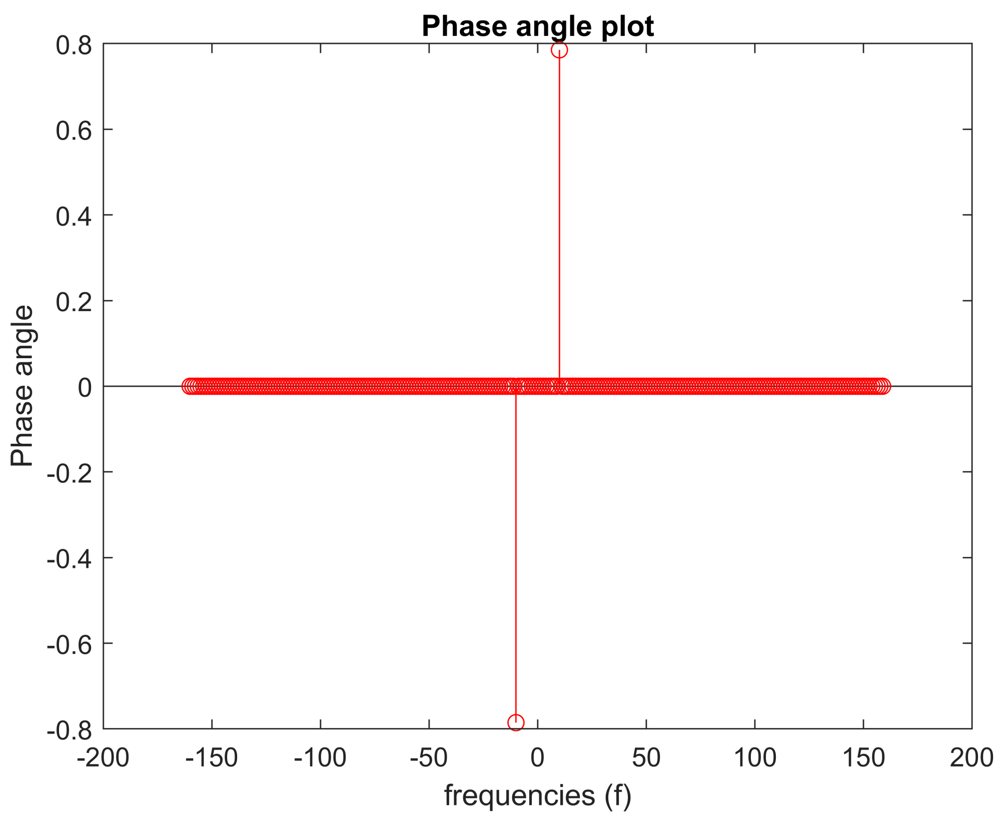
One can use the MATLAB function angle for obtaining the phase value.   angle(H) returns the phase angles, in radians, of a matrix with complex elements. Using this function, it can be noted that the phase angles are calculated and shown for every X values even when they are close to zero.

```
angle_x = angle(X2);
figure;
stem(f,angle_x,'r')
title('Phase angle plot'); xlabel('frequencies (f)'); ylabel('Phase angle');
```

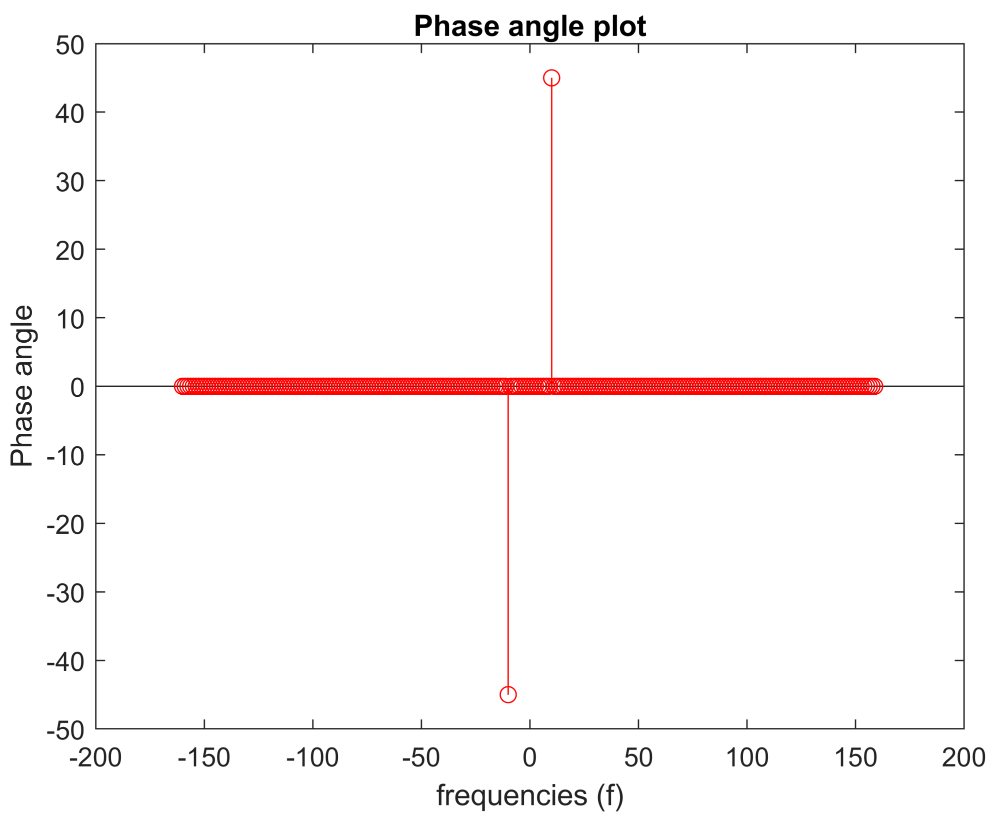


Fixing this requires a tolerance value. If the magnitude is smaller than the tolerance, then we assume a zero phase. This tolerance can be added as follows:

```
tolerance = 0.00001;
X3 = ceil(abs(X2) -tolerance);
X4 = round (X3 ./(X3+1));  %(X4 is the vector of 0s and 1s)
Angle_p = angle(X2).*X4;
figure;
stem(f,Angle_p,'r')
title('Phase angle plot'); xlabel('frequencies (f)'); ylabel('Phase angle');
```

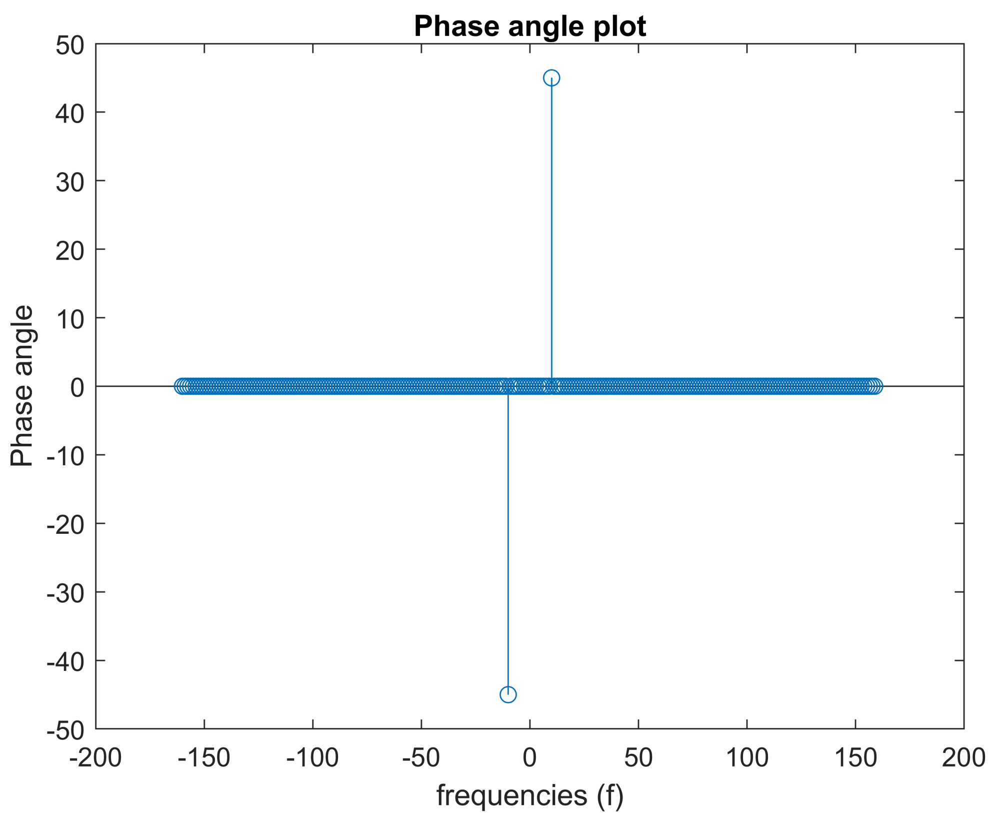


```
% in terms of degrees
Angle_deg = Angle_p*180/pi;
figure;
stem(f,Angle_deg,'r')
title('Phase angle plot'); xlabel('frequencies (f)'); ylabel('Phase angle');
```



Usually function atan2 is used with unwrapping of phase values between $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$, as follows:

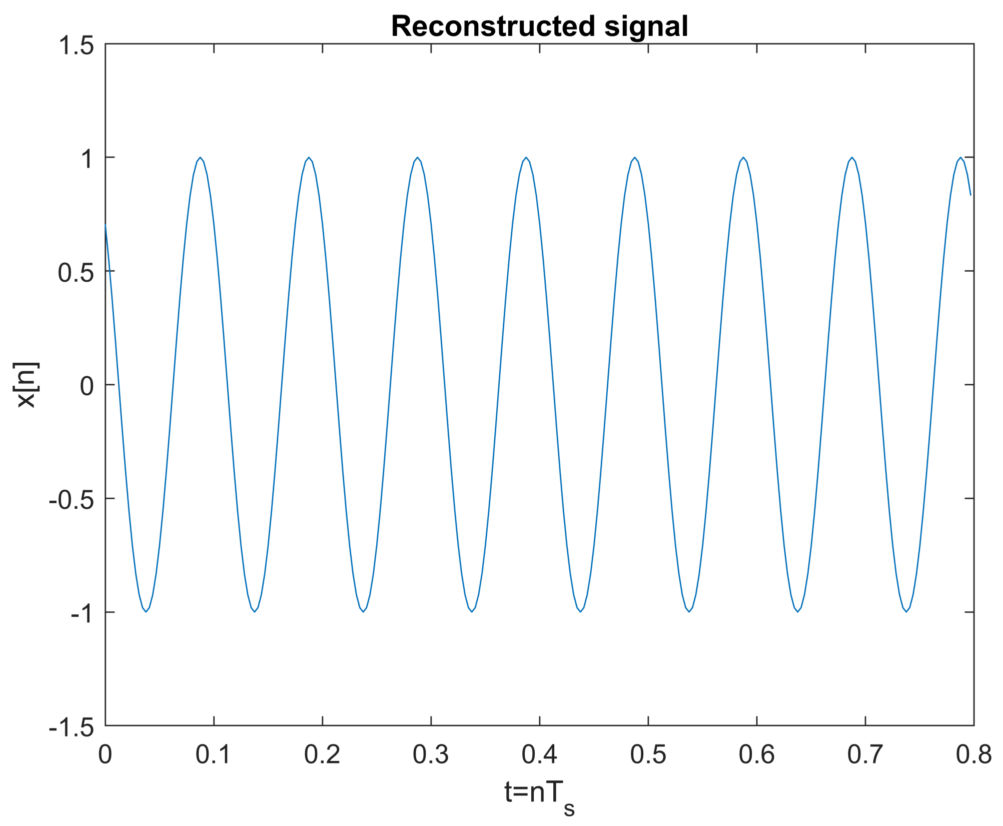
```
X_new=X2;%store the FFT results in another array
%detect noise (very small numbers (eps)) and ignore them
threshold = max(abs(X2))/10000; %tolerance threshold
X_new(abs(X2)<threshold)=0;%maskout values below the threshold
phase=atan2(imag(X_new),real(X_new))*180/pi; %phase information
stem(f,phase); %phase vs frequencies
title('Phase angle plot'); xlabel('frequencies (f)'); ylabel('Phase angle');
```



Reconstructing the time domain signal from the frequency domain samples

Reconstruction of the time domain signal from the frequency domain sample is pretty straightforward. The reconstructed signal, shown below, has preserved the same initial phase shift and the frequency of the original signal. Note: The length of the reconstructed signal is only 256 sample long (0.8 seconds duration), this is because the size of FFT is considered as N = 256. Since the signal is periodic it is not a concern. For more complicated signals, appropriate FFT length (better to use a value that is larger than the length of the signal) need to be used.

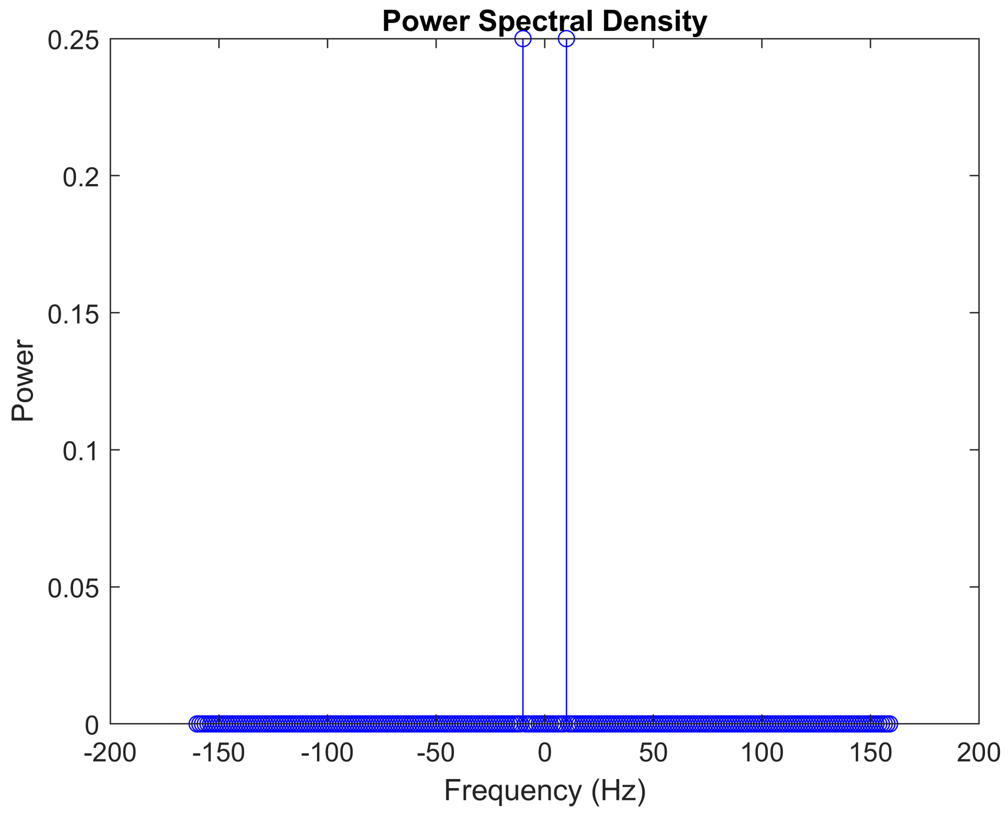
```
x_recon = ifft(ifftshift(X2),N); %reconstructed signal
t = [0:1:length(x_recon)-1]/fs; %recompute time index
plot(t,x_recon);%reconstructed signal
title('Reconstructed signal'); xlabel('t=nT_s'); ylabel('x[n]');
```



Verifying the total power in frequency domain

Here, the total power is verified by applying Discrete Fourier Transform (DFT) on the sinusoidal sequence. The sinusoidal sequence $x[n]$ is represented in frequency domain $X[f]$ using Matlab’s FFT function. The power associated with each frequency point is computed as $P_x[f] = X[f]X^*[f]$

```
Px=X2.*conj(X2)/(N^2); %Power of each freq components
figure
stem(f,Px,'b'); title('Power Spectral Density');
xlabel('Frequency (Hz)');ylabel('Power');
```


Power Spectral Density
0.25
0.2
0.15
0.1
0.05
0
Power
-200
-150
-100
-50
0
50
100
150
200
Frequency (Hz)