[Draft] FFT and IFFT: Visualization with Matlab and Python

Tiep M. H.

I. Fourier Transforms

A. ***Continuous-Time*** *Fourier Transform*

The definition of continuous-time Fourier transform is not unique. Indeed, there are several conventions for defining the Fourier transform. In this material, I will only consider one type of Fourier transform and its inverse Fourier transform. The Fourier transform of a function $f(t)$ is defined as

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt, \tag{1}$$

where $\omega = 2\pi\mu$. On the other hand, the inverse Fourier transform of $F(\omega)$ is defined as

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega = \int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\frac{d(2\pi\mu)}{2\pi} = \int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\mu. \tag{2}$$

B. ***Discrete-Time*** *Fourier Transform (DTFT)*

Considering a **discrete** function $x[k]$, we need a discrete version of Fourier transform. If $x[k]$ is non-zero at $k = \pm\infty$, e.g. $x[k]$ is a periodic signal, then we will use the definition of **discrete-time** Fourier transform (DTFT). To be more specific, the DTFT is defined as

$$F(\omega) = \sum_{k=-\infty}^{\infty} x[k]e^{-j\omega k}. \tag{3}$$

Meanwhile, the inverse DTFT is defined as

$$x[k] = \frac{1}{2\pi}\int_{2\pi} F(\omega)e^{j\omega k}d\omega. \tag{4}$$

In general, the DTFT is still an infinite continuous sequence, where ω is a continuous variable.

*C. **Discrete** Fourier Transform (DFT)*

Similar to the DTFT, we continue to consider a **discrete** function $x[k]$. If $x[k]$ is a *finite* sequence, then we will not need to use the DTFT. Instead, we use the definition of **discrete** Fourier transform (DFT). In particular, the DFT is defined as

$$X[n] = \sum_{k=0}^{K-1} x[k]\ e^{-j(2\pi n)\frac{k}{K}}, \tag{5}$$

where $0 \leq n \leq N-1$ and $N = K$. Then, the inverse DFT is defined as

$$x[k] = \frac{1}{N} \sum_{n=0}^{N-1} X[n]\ e^{j(2\pi k)\frac{n}{N}}, \tag{6}$$

where $0 \leq k \leq K-1$. Obviously, different from the DTFT, the DFT is a finite sequence and has no periodicity. It is not easy to interpret a DFT because the DFT of real data includes complex numbers!

NOTE: *DFT is a special case of the Z-transform, where* $z = e^{j2\pi k/K}$. Moreover, the DFT can be computed efficiently using the fast Fourier transform (FFT).

D. Replacing the DFT by the FFT for efficient computation

It is worth mentioning that the DFT calculation is computationally expensive in practice, while the FFT calculation is much more computationally efficient because it avoids the redundant calculations seen in the DFT. Note that the FFT does **not** define a new Fourier transform, but it is just an efficient algorithm that is used for calculating the DFT. In (5), if we replace K by N, then there are N complex multiplications in calculating $X[n]$. Since we will need to find the sequence $\{X[n]\}_{n=0}^{N-1}$, the total number of complex multiplications we have to perform is N^2. Roughly speaking, the complexity of DFT is $\mathcal{O}(N^2)$. On the other hand, the FFT has the complexity of $\mathcal{O}\left(\frac{N}{2}\log_2(N)\right)$. For example, to evaluate a 1024-point DFT, we need over 1 million complex multiplications! But, we only need over 5000 complex multiplications to evaluate 1024-point FFT.

II. Some Comments on the FFT Algorithm in Matlab/Python

The FFT is a tool for performing the spectral analysis. When employing the FFT, there are a few things we need to know. Below are some terminologies:

- **Frequency bins**: The FFT size is the number of frequency bins. Each bin represents the amount of energy the signal has at that corresponding frequency. Let us denote the FFT size as N.
- **Frequency resolution**: This term is the difference in frequency between each bin. The frequency resolution shows how precise the result will be. By denoting $\Delta_{\text{resolution}}$ as the frequency resolution, we have $\Delta_{\text{resolution}} = (1/N)f_{\text{sampling}}$. Suppose that we sample a signal at $f_{\text{sampling}} = 100$ Hz and use the FFT of size $N = 5$. In this case the frequency resolution is equal to $\Delta_{\text{resolution}} = 100/4 = 25$ Hz. As such, the 1-st freq. bin is at 0 Hz, the 2-nd freq. bin is at 25 Hz, the 3-rd freq. bin is at 50 Hz, the 4-th freq. bin is at 75 Hz, and the 5-th freq. bin is at 100 Hz. Now, if the signal has the frequency of 66 Hz, then the energy will be between the 3-rd freq. bin (50 Hz) and the 4-th freq. bin (75 Hz). With $\Delta_{\text{resolution}} = 25$ Hz, we can see that the the measurement will be not quite accurate.

Let us rewrite the DFT in (5) as follows:

$$X[n] = \sum_{k=0}^{K-1} x[k]\ e^{-j(2\pi n)\frac{k}{K}} = \sum_{k=0}^{K-1} x[k]\ e^{-j\widehat{\omega_n} k} \tag{7}$$

where $\widehat{\omega_n}$ is defined as

$$\widehat{\omega_n} = \frac{2\pi n}{K}.$$

It is obvious to see that $\widehat{\omega_n}$ is a **non-negative** value, i.e. $\widehat{\omega_n} \geq 0$. However, using Euler's formula, we can rewrite

$$\begin{aligned} e^{-j\widehat{\omega_n} k} &= \begin{cases} cos(\widehat{\omega_n} k) + 1j \times sin(\widehat{\omega_n} k), & \text{if } 0 \leq n \leq \frac{K}{2}, \\ cos(\widehat{\omega_n} k) + 1j \times sin(\widehat{\omega_n} k) = cos(\widehat{\omega_{N-n}} k) + 1j \times sin(-\widehat{\omega_{N-n}} k), & \text{if } \frac{K}{2} < n \leq K-1; \end{cases} \\ &= cos(\omega_n k) + 1j \times sin(\omega_n k) \\ &= e^{-j\omega_n k}, \end{aligned} \tag{8}$$

where ω_n is defined as

$$\omega_n = \begin{cases} \widehat{\omega_n}, & \text{if } 0 \leq n \leq \frac{K}{2}, \\ -\widehat{\omega_{N-n}}, & \text{if } \frac{K}{2} < n \leq K-1. \end{cases} \tag{9}$$

It is seen that ω_n can take a negative value, thus we will use ω_n rather than $\widehat{\omega_n}$.

For example, let us consider $k = 1$ and $K = N = 16$. Using (9), we have $\omega_1 = \frac{2\pi\times 1}{16} = \frac{\pi}{8}$ at $n = 1$. Similarly, at $n = 5$, we have $\omega_5 = \frac{2\pi\times 5}{16} = \frac{5\pi}{8}$. With $n = 14$ within the range between $\frac{K}{2}$ and $(K-1)$, we rely on (9) to calculate $\omega_{14} = -\frac{2\pi(16-14)}{16} = -\frac{\pi}{4} < 0$. The following figure illustrates the relationship between the index n and the quantity ω_n in the example.

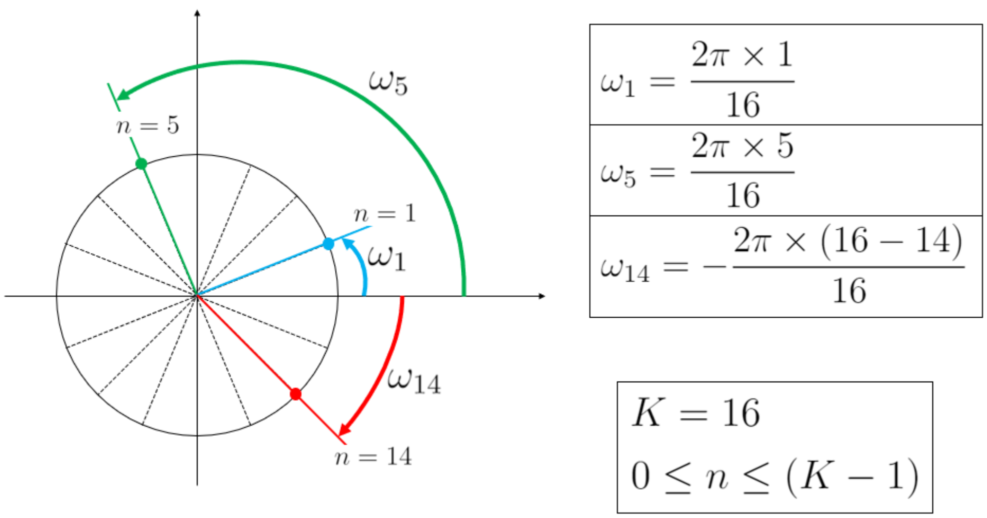

Fig. 1. The relationship between n and ω_n.

Let us look at another example, where we set $N = K = 2^7 = 128$. To perform the FFT algorithm in Matlab, we use the `fft` function. The results returned by the `fft` function are $F[0], F[1], \ldots, F[N-1]$. Portraying the amplitude $|X[n]|$ vs the index n, we obtain Figure II as can be seen below. When n increases, we go through the following domains:

low positive freq $\rightarrow$ high positive freq $\rightarrow$ high negative freq $\rightarrow$ low negative freq.

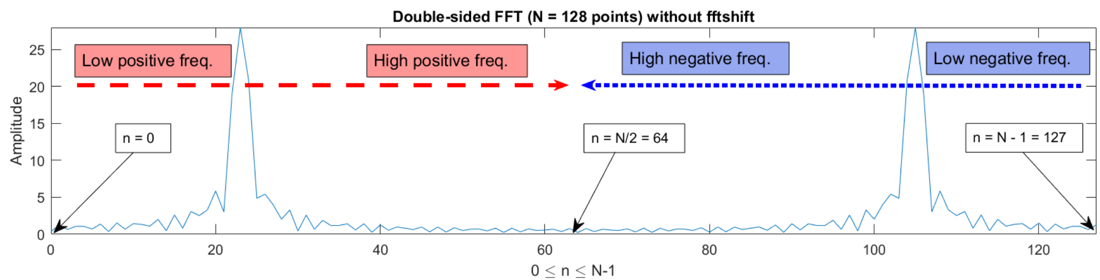

Fig. 2. The amplitude $|X[n]|$ versus the index n after using the `fft` function to perform the DFT. Note that until this point, the `fftshift` function has not yet been invoked.

Since we want to depict the amplitude $|X[n]|$ vs the frequency f_n, we will have to rearrange the x-axis so that f_n goes through the following domains:

high negative freq $\rightarrow$ low negative freq $\rightarrow$ low positive freq $\rightarrow$ high positive freq.

To perform this rearrangement, we apply the `fftshift` function to the result of the `fft` function. Note that, the new x-axis will range from $\left(-\frac{K}{2}\right) \times \Delta_{\text{resolution}}$ to $\left(\frac{K}{2} - 1\right) \times \Delta_{\text{resolution}}$, where $\Delta_{\text{resolution}} = \frac{f_{\text{sampling}}}{K}$ is the frequency resolution. The following figure is the result obtained by using the `fftshift` function and the new x-axis.

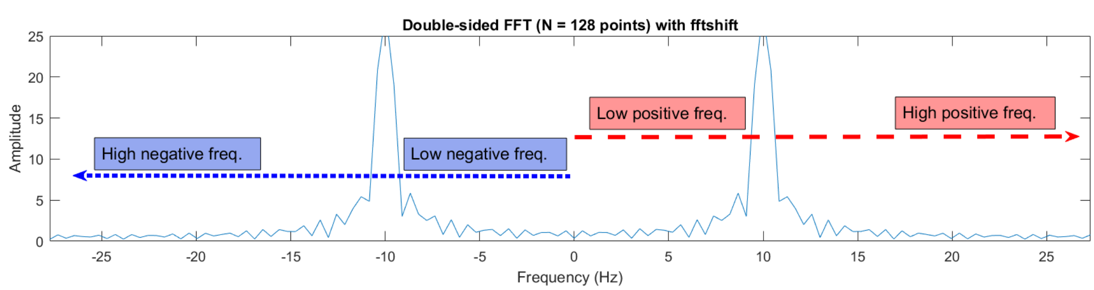

Fig. 3. The amplitude $|X[n]|$ versus the frequency f_n.

III. Visualization of the FFT Algorithm

A. FFT Algorithm: Visualization with Matlab

```
clear all; clc;
%% Time-discrete signal
f = 10;
fs = 5*f; % sampling freq. >= 2 max signal freq. (according to Nyquist theorem)
phase = (1/3)*pi;
t = 0:1/fs:1; % time base
x = sin(2*pi*f*t + phase);

figure()
plot(t, x, '--o');
title(['Sine wave with f = ', num2str(f), ' Hz']);
xlabel('Time (s)');
ylabel('Amplitude');

%% Perform the DFT through the use of the FFT algorithm
figure()
subplot(3, 1, 1);
N = 2^5; % consider N-point DFT
delta = fs/N; % freq. resolution
freqs = (-N/2:N/2-1) * delta; % negative and positive frequencies
X = fftshift(fft(x, N)); % compute DFT using FFT
%
plot(freqs, abs(X)); % NOTE: we only take the amplitude of X_n
title(['Double-sided FFT with N = ', num2str(N), ' points']);
xlabel('Frequency (Hz)')
ylabel('Amplitude');
ylim([0, 25])

```

```
hold on

subplot(3, 1, 2);
N = 2^7; % consider N-point DFT
delta = fs/N; % freq. resolution
freqs = (-N/2:N/2-1) * delta; % negative and positive frequencies
X = fftshift(fft(x, N)); % compute DFT using FFT
%
plot(freqs, abs(X)); % NOTE: we only take the amplitude of X_n
title(['Double-sided FFT with N = ', num2str(N), ' points']);
xlabel('Frequency (Hz)')
ylabel('Amplitude');
ylim([0, 25])

subplot(3, 1, 3);
N = 2^9; % consider N-point DFT
delta = fs/N; % freq. resolution
freqs = (-N/2:N/2-1) * delta; % negative and positive frequencies
X = fftshift(fft(x, N)); % compute DFT using FFT
%
plot(freqs, abs(X)); % NOTE: we only take the amplitude of X_n
title(['Double-sided FFT with N = ', num2str(N), ' points']);
xlabel('Frequency (Hz)')
ylabel('Amplitude');
ylim([0, 25])
```

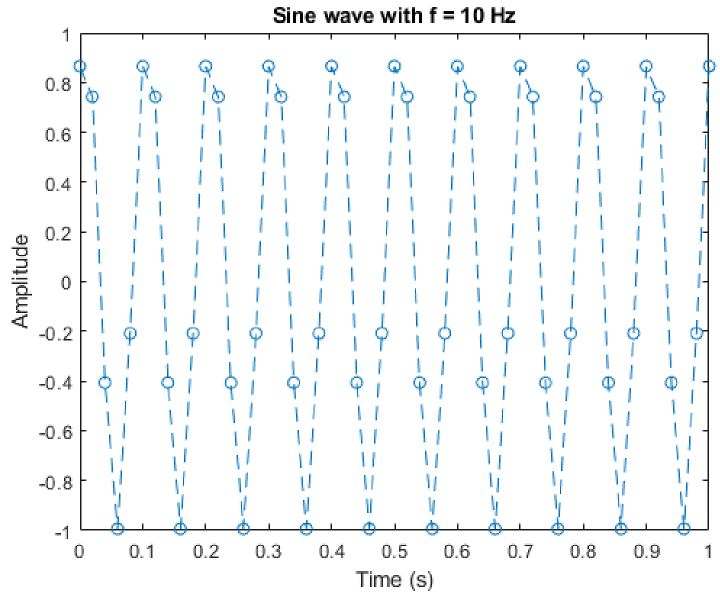

Fig. 4. Visualization with Matlab. The function $sin(2\pi ft + \pi/3)$ is sampled at $f_s = 50$ Hz.

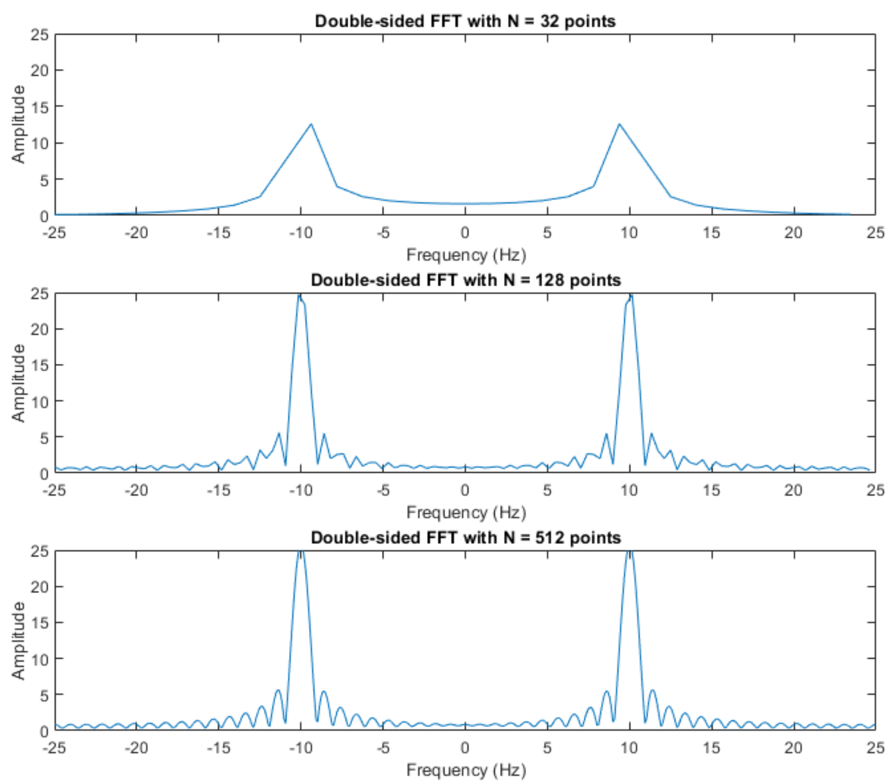

Fig. 5. Visualization with Matlab. For $N \in \{2^5, 2^7, 2^9\}$, the double-sided FFT (with fftshift) is illustrated. The result becomes more accurate when N increases.

B. FFT Algorithm: Visualization with Python

```python
import numpy as np
from scipy.fft import fft, fftshift
import matplotlib.pyplot as plt

""" Time-discrete signal """
f = 10
fs = 5*f  # sampling freq. >= 2 max signal freq. (according to Nyquist theorem)
phase = (1/3)*np.pi
t = np.array([i*(1/fs) for i in range(fs+1)])
x = np.sin(2*np.pi*f*t + phase)

plt.figure()
plt.plot(t, x, '--o')
plt.title("Sine wave with f = {:2.0f}".format(f) + ' Hz')
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("Amplitude", fontsize=12)

""" Perform the DFT through the use of the FFT algorithm """
fig, axs = plt.subplots(3, figsize=(4, 6), sharey=True)
fig.subplots_adjust(hspace=0.5)  # adjust the spacing

N = 2**5
delta = fs/N
freqs = np.array([(-N/2 + i)*delta for i in range(N)])
X = fftshift(fft(x, N))
#
axs[0].plot(freqs, abs(X), '-')
axs[0].set_title("Double-sided FFT with N = %2.0f" % N + ' points')
axs[0].set_xlabel('Frequency (Hz)')
axs[0].set_ylabel('Amplitude')
axs[0].set_xlim([freqs[0], freqs[-1]])
axs[0].set_ylim([0, 25])

N = 2**7
delta = fs/N
freqs = np.array([(-N/2 + i)*delta for i in range(N)])
X = fftshift(fft(x, N))
#
axs[1].plot(freqs, abs(X), '-')
axs[1].set_title("Double-sided FFT with N = %2.0f" % N + ' points')
axs[1].set_xlabel('Frequency (Hz)')
axs[1].set_ylabel('Amplitude')
axs[1].set_xlim([freqs[0], freqs[-1]])
```

```
axs[1].set_ylim([0, 25])

N = 2**9
delta = fs/N
freqs = np.array([(-N/2 + i)*delta for i in range(N)])
X = fftshift(fft(x, N))
#
axs[2].plot(freqs, abs(X), '-')
axs[2].set_title("Double-sided FFT with N = %2.0f" % N + '
 ↪  points')
axs[2].set_xlabel('Frequency (Hz)')
axs[2].set_ylabel('Amplitude')
axs[2].set_xlim([freqs[0], freqs[-1]])
axs[2].set_ylim([0, 25])

fig.tight_layout()
```

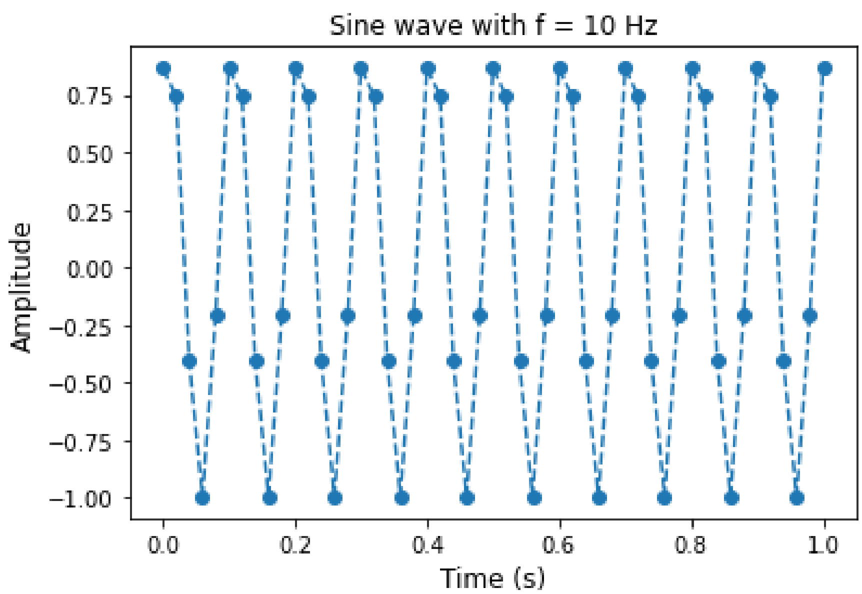

Fig. 6. Visualization with Python. The function $sin(2\pi ft + \pi/3)$ is sampled at $f_s = 50$ Hz.

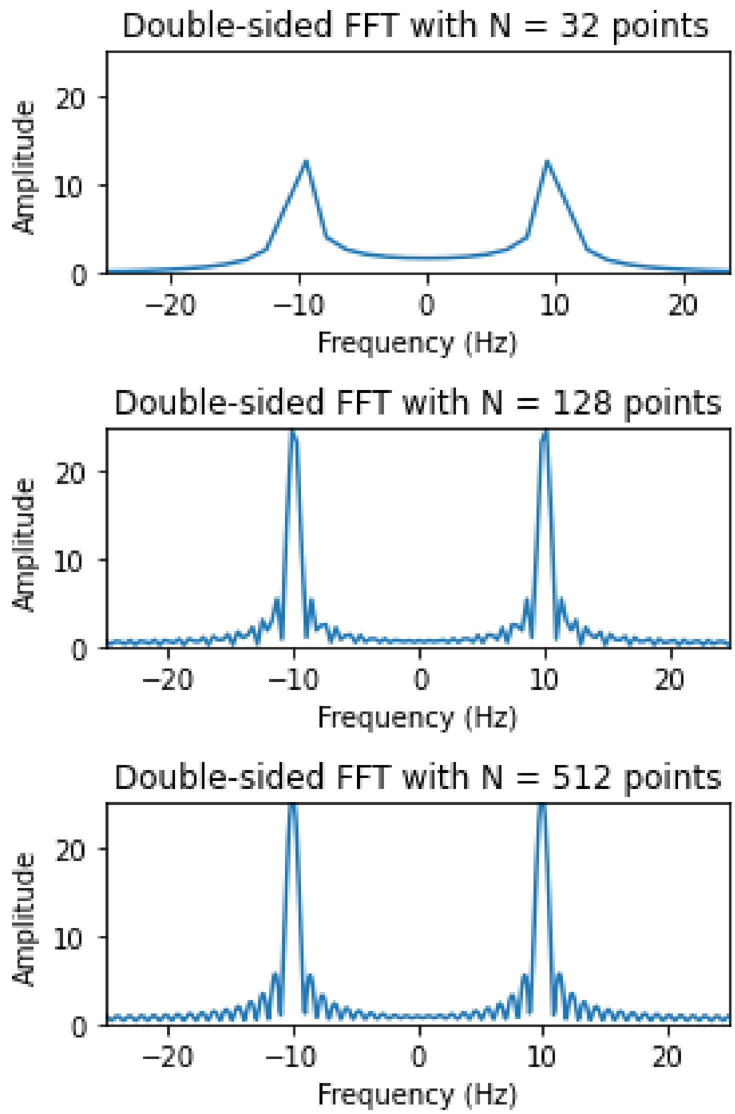

Fig. 7. Visualization with Python. For $N \in \{2^5, 2^7, 2^9\}$, the double-sided FFT (with fftshift) is illustrated. The result becomes more accurate when N increases.

IV. Power Spectral Density (PSD)

The *power spectral density* (PSD) describes how power is distributed over a range of frequencies. It can be derived from the FFT magnitude (aka the DFT magnitude) as follows:

$$P[n] = \frac{1}{N}\left|X[n]\right|^2, \; 0 \le n \le N-1, \tag{10}$$

where $\{X[n]\}_{n=0}^{N-1}$ is a sequence of N values obtained by using the FFT. Let us assume that N is divisible by 2. The total power is equal to

$$P_{\text{total}} = \underbrace{P[0] + \underbrace{P[1] + \ldots + P[N/2-1]}_{\text{Part A}} + P[N/2]}_{\text{0 Hz (at n=0) + Positive frequencies}} + \underbrace{P[N/2+1] + \ldots + P[N-1]}_{\text{Part B}}. \tag{11}$$

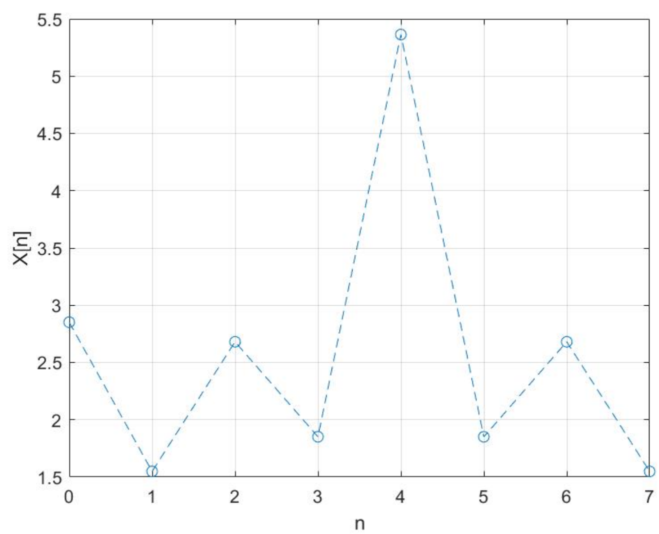

Fig. 8. For $N = 2^3 = 8$, we can see that $X[1] = X[8-1]$, $X[2] = X[8-2]$, $X[3] = X[8-3]$. Thus, Part A in (11) is exactly the same as Part B in (11).

Looking at the illustration in Fig 8, we can the resemblance between Part A and Part B. This resemblance is also seen in Figs 2 and 3. As such, we can take the first $N/2+1$ values in the sequence $\{X[n]\}_{n=0}^{N-1}$ to calculate a new quantity $P^{\text{first}}[\widehat{n}]$ as follows:

$$P^{\text{first}}[\widehat{n}] = P[0] \text{ where } \widehat{n} = 0, \tag{12a}$$

$$P^{\text{first}}[\widehat{n}] = {\color{red}2} \times \frac{1}{N}\left|X[\widehat{n}]\right|^2, \text{ where } \underbrace{1 \le \widehat{n} \le N/2-1}_{\color{red}\text{Part A}}, \tag{12b}$$

$$P^{\text{first}}[\widehat{n}] = P[N/2] \text{ where } \widehat{n} = N/2. \tag{12c}$$

Note that we have the factor of 2 in (12b) because we want to **conserve the total power**, i.e.

$$\boxed{P_{\text{total}} = P[0] + \underbrace{P^{\text{first}}[1] + \ldots + P^{\text{first}}[N/2-1]}_{\color{red}\text{two times of Part A}} + P[N/2]. \quad (13)}$$

We can also *normalize* the PSD by dividing the above results by the sampling frequency. Then, we can replace (10) and (12) by the following expression:

$$P[n] = \frac{(10)}{f_{\text{sampling}}}, \; 0 \le n \le N-1, \tag{14}$$

$$P^{\text{first}}[\widehat{n}] = \frac{(12)}{f_{\text{sampling}}}, \; 0 \le \widehat{n} \le N/2. \tag{15}$$

If the unit of measurement in (10) and (12) is Watt, then the unit of measurement in (14) and (15) is Watt/Hz.

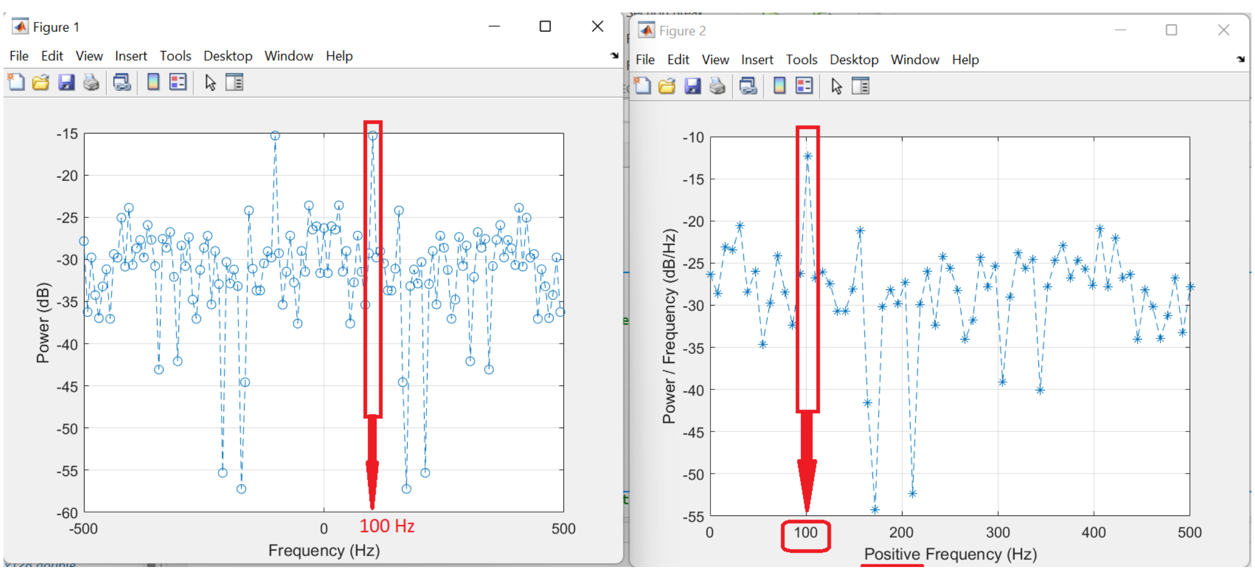

Fig. 9. Either (14) or (15) is used, we can estimate the carrier frequency is equal to 100 Hz.

V. Visualization of PSD Estimate

A. PSD Estimate: Visualization with Matlab

```matlab
%% In this toy example, let us look at a method of estimating PSD
% Power Spectral Density = PSD

clear all; clc;
%% Time-discrete signal
f = 100;
fs = 10*f; % sampling freq.
t = 0:1/fs:1;
x = cos(2*pi*f*t) + randn(size(t));

%% Perform the DFT through the FFT algo.
N = 2^7; % consider N-point DFT
delta = fs/N;
freqs = (-N/2:N/2-1) * delta;
X = fft(x, N); % N-point DFT

%% PSD Estimate with double-sided FFT
X_swapped = fftshift(X);
PSD = (1/(fs*N)) * abs(X_swapped).^2; % we rely on the double-sided FFT

figure()
plot(freqs, 10*log10(PSD), '--o')
xlabel('Frequency (Hz)')
ylabel('Power (dB)')
hold on
grid on

```

```
%% Take the first (N/2 + 1) points (corres. to 0 Hz + positive
   freqs)
X_FirstHalf = X(1:N/2+1); % the first (N/2 + 1) points
freqs_positive = (0:N/2) * delta; % 0 Hz + positive frequencies

%% PSD Estimate with one-sided FFT
PSD_FirstHalf = (1/(fs*N)) * abs(X_FirstHalf).^2;
PSD_FirstHalf(2:N/2) = 2 * PSD_FirstHalf(2:N/2); % 2 times of
   Part A

figure()
plot(freqs_positive, 10*log10(PSD_FirstHalf), '--*')
grid on
xlabel('Positive Frequency (Hz)')
ylabel('Power / Frequency (dB/Hz)')

%% Check the total power
sum(PSD)
sum(PSD_FirstHalf)
sum(PSD)/sum(PSD_FirstHalf)
```

VI. References

[1] https://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf

[2] https://mathworld.wolfram.com/Z-Transform.html

[3] http://www.add.ece.ufl.edu/4511/references/ImprovingFFTResoltuion.pdf

[4] https://blog.endaq.com/vibration-analysis-fft-psd-and-spectrogram