SIGNAL PROCESSING SYSTEM DESIGN LAB

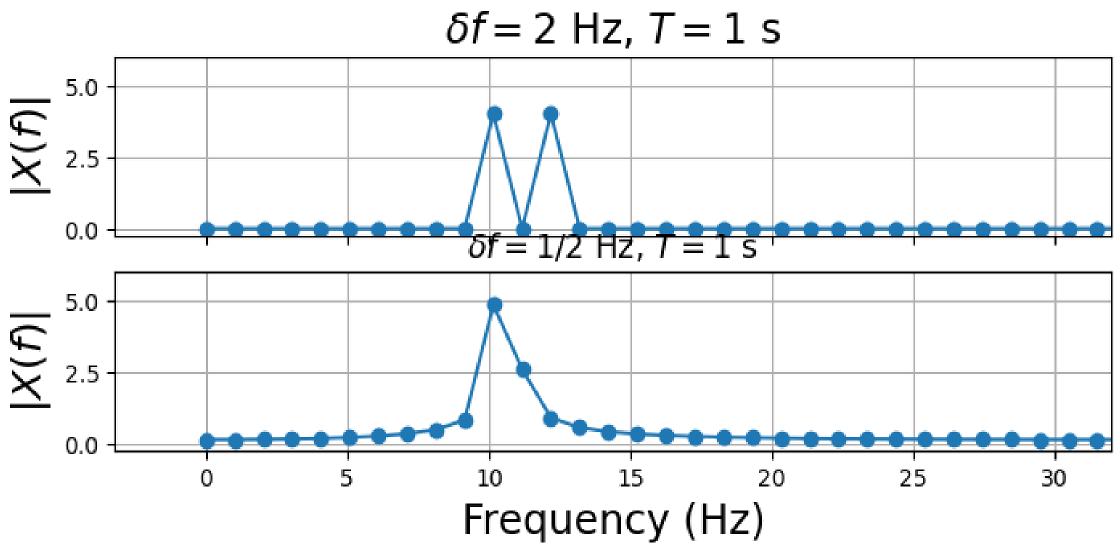
YASH ANAND
(23EE65R22)

4. Introducing Spectral Analysis

In this section, I have explained the shape of windows functions in terms of spectral leakage and developed the concept of processing gain and equivalent noise bandwidth as closely related metrics for categorizing windows.
The window functions are implemented in the scipy.signal.windows submodule
Window functions are also fundamental to antenna analysis for many of the same reasons,especially with respect to linear arrays.
I illustrated the major issues involved in using window functions for spectral analysis and derived the most common figures of merit for some popular windows.

4.1
AIM:
To illustrate the effect of frequency difference on the DFT and how the frequency difference affects the amplitude of the peak in the DFT.
FUNCTION IMPORTED:
numpy.fft.fft(): This function performs the discrete Fourier transform of a signal.
numpy.linspace(): This function creates a evenly spaced array of numbers.
plt.subplots(): This function creates a subplots object, which is a collection of axes objects.
plt.plot(): This function plots a line or curve.
plt.title(): This function sets the title of the plot.
plt.ylabel(): This function sets the y-axis label of the plot.
plt.xlabel(): This function sets the x-axis label of the plot.
plt.grid(): This function adds a grid to the plot.

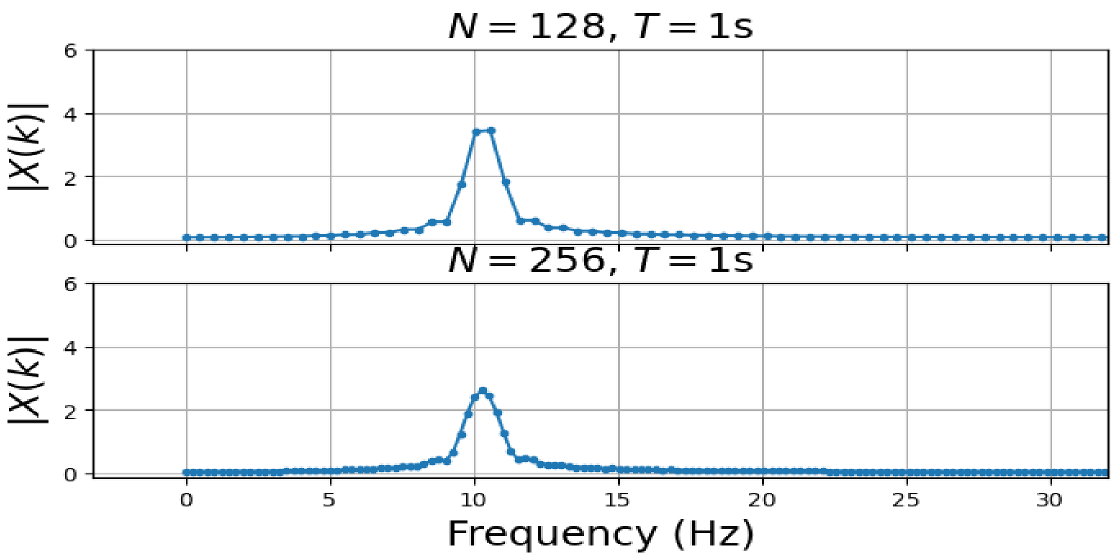
OBSERVATION:
I have plotted the magnitude of the DFT for the sum of two equal-amplitude tones separated by 2Hz. Using the parameters I have chosen for the DFT, I can easily see there are two distinct frequencies in the input signal. However, in the bottom plot, the same tones are only separated by 0.5Hz and cannot be distinguished in the DFT.
OUTPUT:



4.2
AIM:

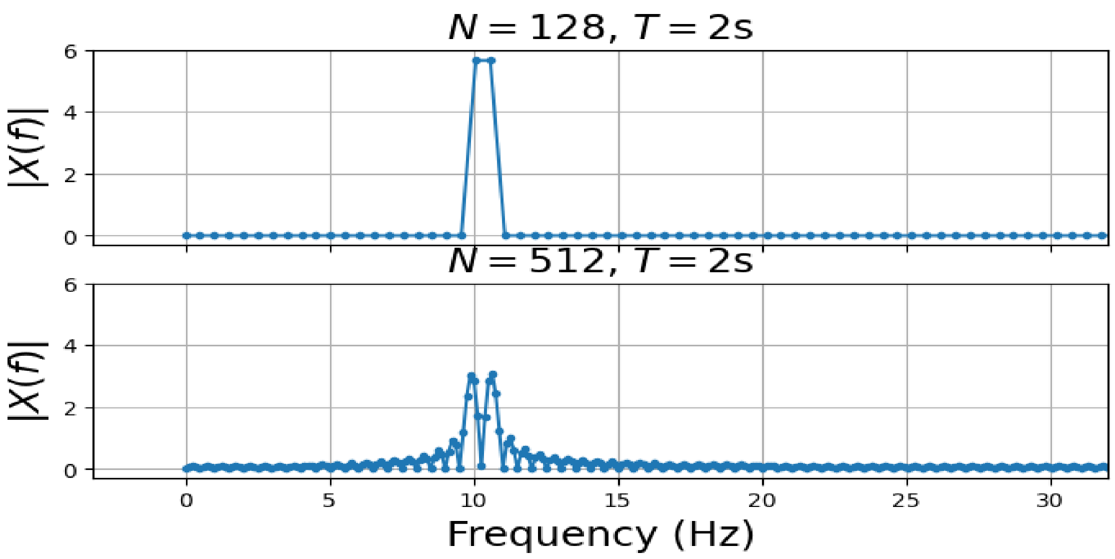
To illustrate the effect of DFT size on the resolution of the DFT.The DFT size is the number of samples used to compute the DFT. A larger DFT size results in a higher resolution DFT, which means that the DFT can resolve more closely spaced frequencies.
FUNCTION IMPORTED:
numpy.fft.fft(): This function performs the discrete Fourier transform of a signal.
numpy.linspace(): This function creates a evenly spaced array of numbers.
plt.subplots(): This function creates a subplots object, which is a collection of axes objects.
plt.plot(): This function plots a line or curve.
plt.title(): This function sets the title of the plot.
plt.ylabel(): This function sets the y-axis label of the plot.
plt.xlabel(): This function sets the x-axis label of the plot.
plt.grid(): This function adds a grid to the plot.

OBSERVATION:
The observation from the code is that the larger the DFT size, the better the resolution of the DFT. This is because a larger DFT size allows the DFT to resolve more closely spaced frequencies.
OUTPUT:



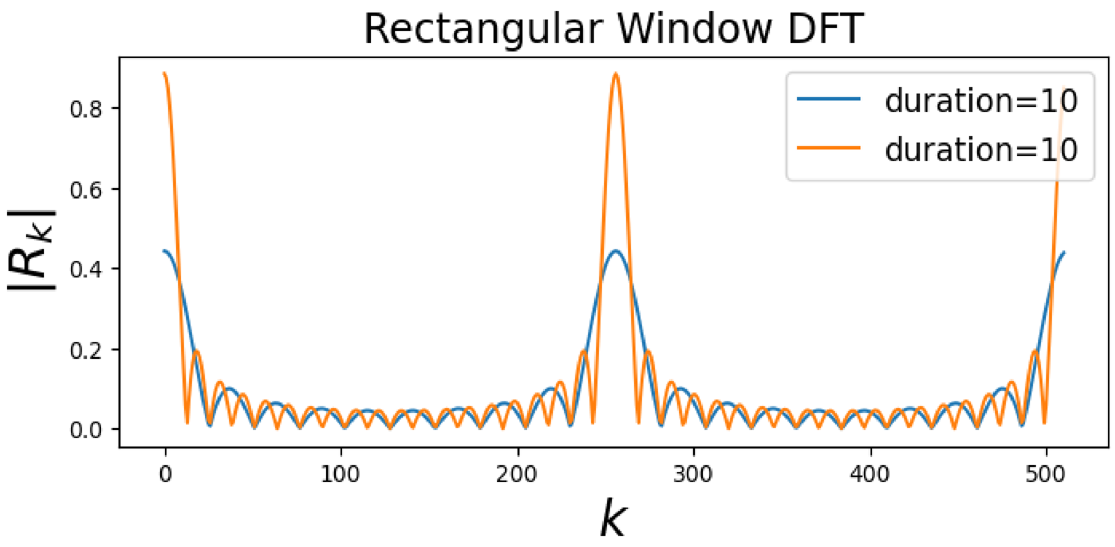
4.3
AIM:
To illustrate the effect of the sampling frequency on the resolution of the DFT and to plots the discrete Fourier transform (DFT) of a signal for two different values of the sampling frequency FUNCTION IMPORTED:
numpy.fft.fft(): This function performs the discrete Fourier transform of a signal.
numpy.linspace(): This function creates a evenly spaced array of numbers.
plt.subplots(): This function creates a subplots object, which is a collection of axes objects.
plt.plot(): This function plots a line or curve.
plt.title(): This function sets the title of the plot.
plt.ylabel(): This function sets the y-axis label of the plot.
plt.xlabel(): This function sets the x-axis label of the plot.
plt.grid(): This function adds a grid to the plot.

OBSERVATION:
The observation from the code is that the higher the sampling frequency, the better the resolution of the DFT. This is because a higher sampling frequency allows the DFT to resolve more closely spaced frequencies.
OUTPUT:



4.4
AIM:
To illustrate the effect of the window duration on the DFT. FUNCTION IMPORTED:
the function abs sinc() is defined to calculate the absolute value of the sinc function. The function takes three arguments:
k: The index of the DFT coefficient.
N: The number of DFT coefficients.
Ns: The window duration.

OBSERVATION:
The observation from the code is that the longer the window duration, the smoother the DFT becomes. However, the longer the window duration, the lower the resolution of the DFT becomes.
OUTPUT:

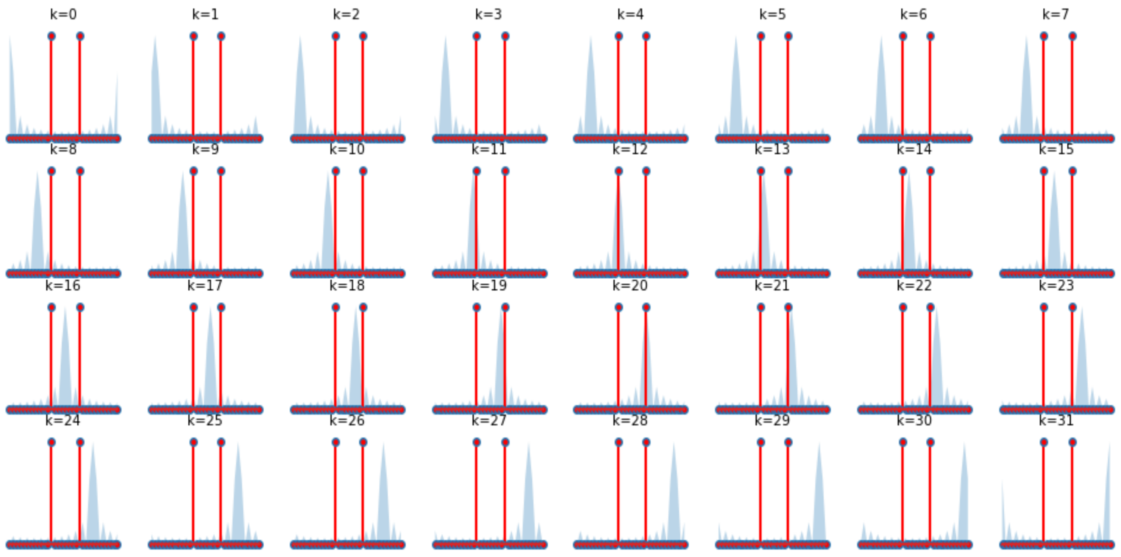


4.5
AIM:
To illustrate the effect of the window function on the DFT.
FUNCTION IMPORTED:
the function dftmatrix() is defined to construct the DFT matrix. The function takes two arguments:
Nfft: The size of the DFT.
N: The length of the signal.
The function plt.fill_between() is used to fill a region between two curves.

OBSERVATION:

The observation from the code is that the rectangular window has a significant effect on the DFT. The DFT of the signal with the rectangular window is smoother than the DFT of the original signal. However, the DFT of the signal with the rectangular window also has lower resolution than the DFT of the original signal.

OUTPUT:



4.6

AIM:

To illustrate the effect of the window function on the DFT.

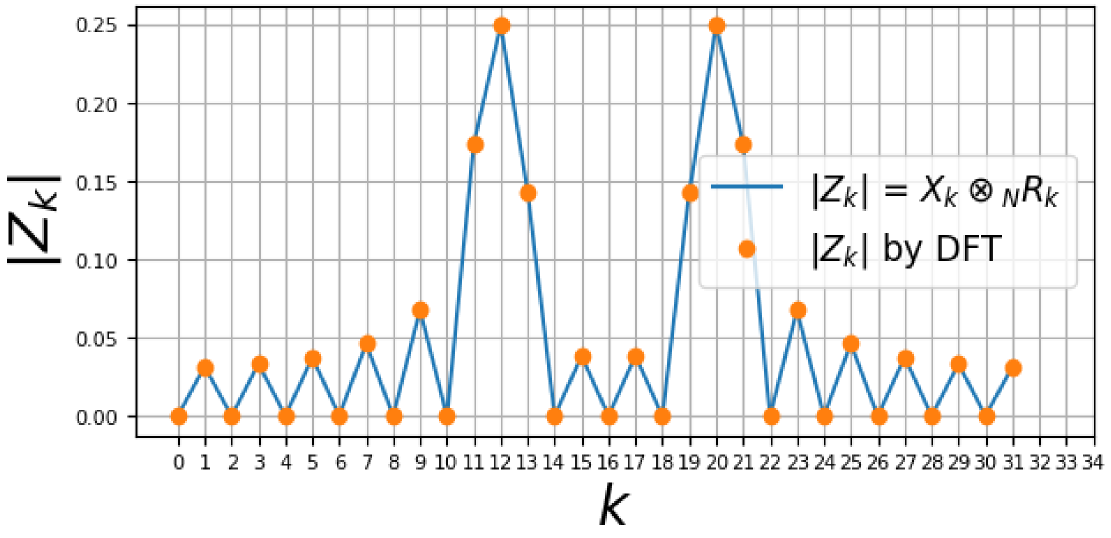
FUNCTION IMPORTED:

The function abs([idx,0]*X)/np.sqrt(Nf) calculates the DFT of the product of the signal and the window function. The function abs(Z) calculates the DFT of the signal.

OBSERVATION:

The observation from the code is that the DFT of the product of the signal and the window function is similar to the DFT of the signal, but it is smoother. This is because the window function has a non-zero width, which spreads out the energy of the signal over a wider range of frequencies.

OUTPUT:



4.7

AIM:

To illustrate the effect of the amplitude of the signal on the DFT.

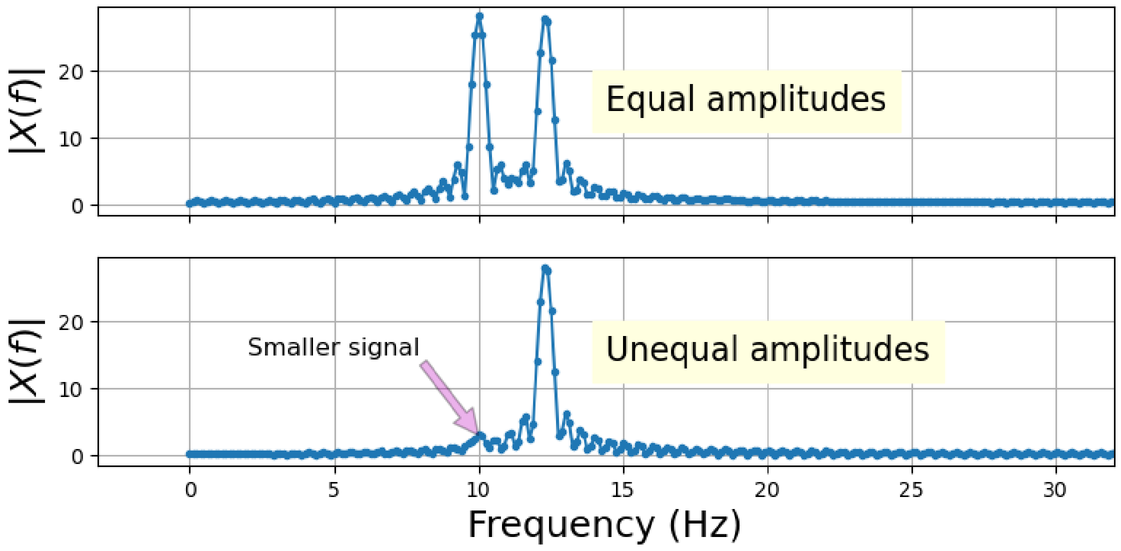
FUNCTION IMPORTED:

dftmatrix(): This function constructs the DFT matrix.

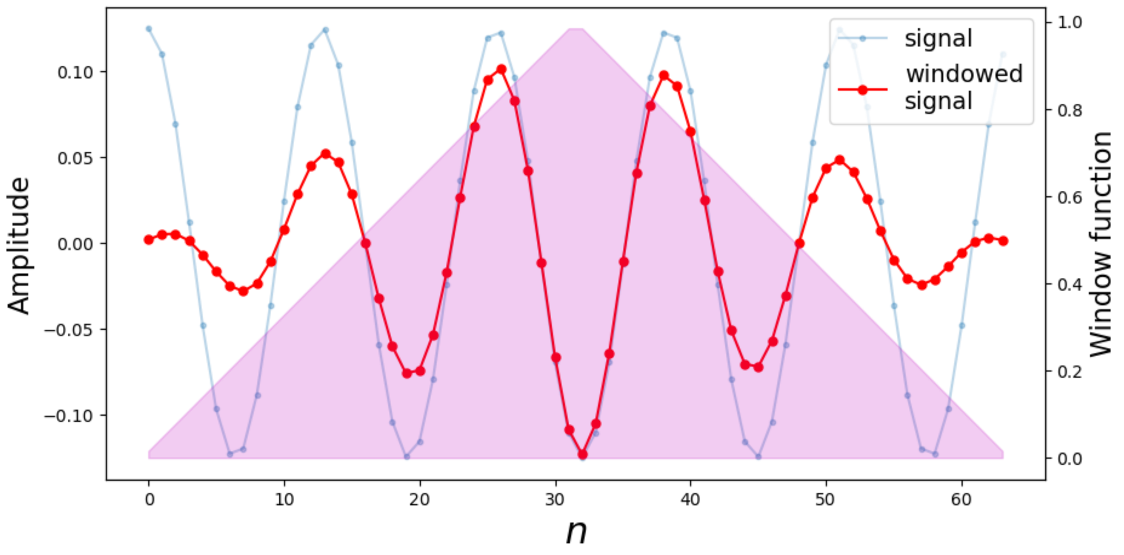
db20(): This function converts the DFT to decibels.

OBSERVATION:
The observation from the code is that the DFT of the signal with equal amplitudes has two peaks, one at the frequency of the first signal and the other at the frequency of the second signal. The DFT of the signal with one signal having 10 times the amplitude of the other signal has a larger peak at the frequency of the louder signal.

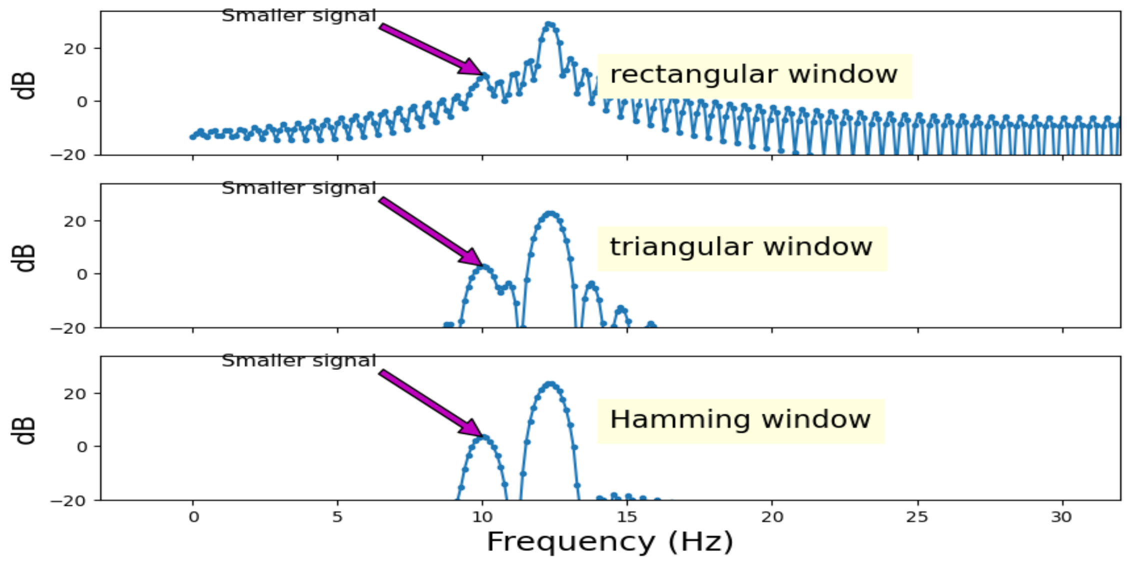
The reason for this is that the DFT of a signal is proportional to the amplitude of the signal. So, if one signal has a larger amplitude than the other signal, the DFT of the louder signal will have a larger peak.
OUTPUT:



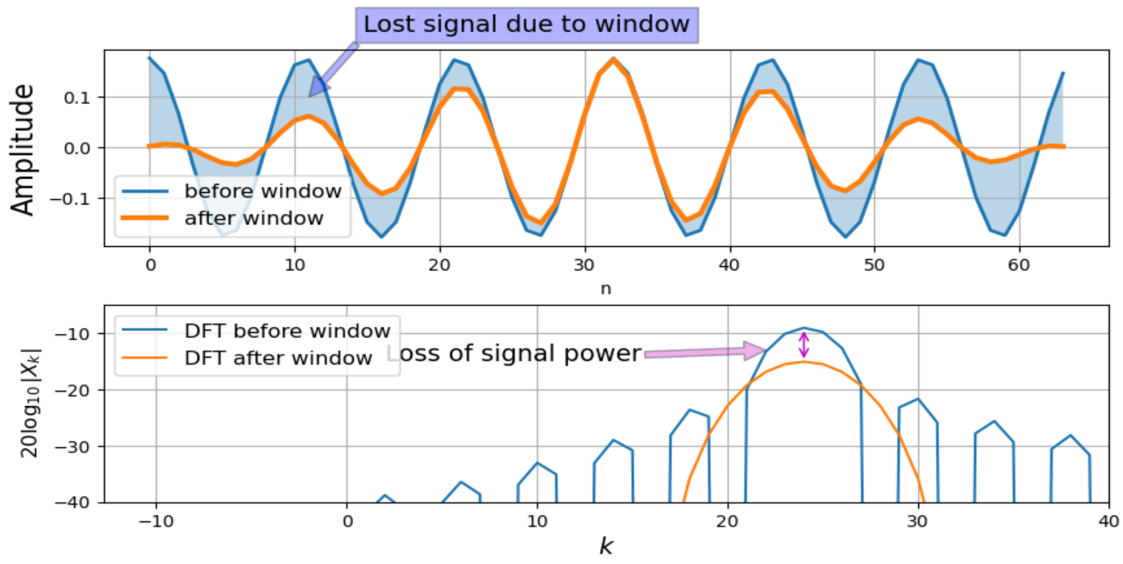
4.8
AIM:
To illustrate the effect of the window function on the signal.
FUNCTION IMPORTED:
signal.triang(): This function creates a triangular window.
OBSERVATION:
The observation from the code is that the windowed signal has a smoother appearance than the original signal. This is because the window function is applied to the signal before the DFT is computed. The window function has a non-zero width, which spreads out the energy of the signal over a wider range of frequencies.
The reason for this is that the DFT of a signal is proportional to the energy of the signal. So, if the energy of the signal is spread out over a wider range of frequencies, the DFT will be smoother.
Here the windowed signal has a smoother appearance than the original signal because the triangular window has a non-zero width of 0.5. This means that the energy of the signal is spread out over a wider range of frequencies.
OUTPUT:



4.9
AIM:
To illustrate the effect of the window function on the DFT.
FUNCTION IMPORTED:
signal.triang(): This function creates a triangular window.
signal.hamming(): This function creates a Hamming window.
OBSERVATION:
The observation from the code is that the DFT of the signal with the rectangular window has sidelobes that are much larger than the DFT of the signal with the triangular or Hamming windows. The sidelobes are the smaller peaks around the main peak in the DFT.
The reason for this is that the rectangular window has a non-zero width of 1. This means that the energy of the signal is spread out over a wider range of frequencies, which results in larger sidelobes.
The triangular and Hamming windows have a smaller width than the rectangular window. This means that the energy of the signal is spread out over a narrower range of frequencies, which results in smaller sidelobes. OUTPUT:

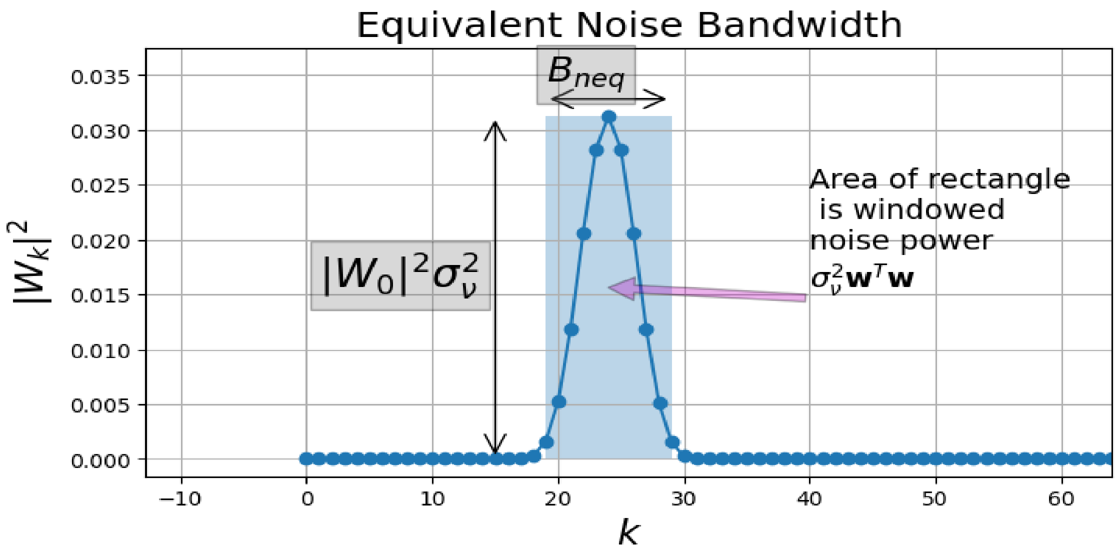


4.10
AIM:
To plots the time domain and frequency domain of a signal before and after applying a window function and to illustrate the effect of the window function on the signal.
FUNCTION IMPORTED:
dftmatrix(): This function constructs the DFT matrix.
db20(): This function converts the DFT to decibels.
OBSERVATION:
The observation from the code is that the window function has a significant effect on the signal. The signal after applying the window function has a lower amplitude and a narrower bandwidth. The reason for this is that the window function is multiplied by the signal before the DFT is computed. The window function has a non-zero width, which spreads out the energy of the signal over a wider range of frequencies.
When the window function is multiplied by the signal, the energy of the signal is spread out over a narrower range of frequencies.
OUTPUT:



4.11
AIM:
To plots the power spectrum of a signal after applying a window function with the aim of understanding the concept of equivalent noise bandwidth.
FUNCTION USED:
abs(fft.fft(np.array(np.diag(w) * u).flatten(), N) / np.sqrt(N))**2: This line calculates the power spectrum of the windowed signal.
ax.add patch(Rectangle((idx - 10 / 2, 0), width=10, height=Xm[idx], alpha=0.3)): This line adds a rectangle to the plot. The rectangle represents the noise power introduced by the window function.

OBSERVATION:
The observation from the code is that the area of the rectangle is equal to the noise power introduced by the window function. The noise power is proportional to the square of the amplitude of the window function. The width of the rectangle is equal to the equivalent noise bandwidth OUTPUT:

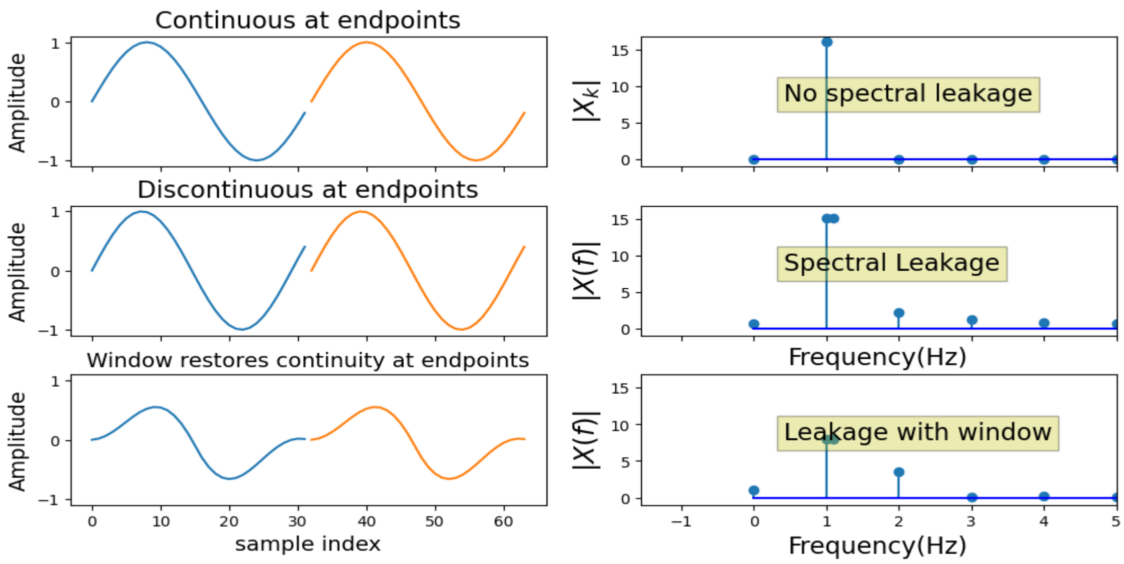


4.12
AIM:
To plots the time and frequency domain representations of a signal with different types of discontinuities at the end-points with the aim of the code is to illustrate the effect of spectral leakage on the frequency domain representation of a signal.

FUNCTION IMPORTED:
signal.triang(): This function creates a triangular window.
np.fft.fft(): This function computes the discrete Fourier transform of a signal.

np.abs(): This function calculates the absolute value of a number or array.
np.stem(): This function plots a stem plot of a signal.

OBSERVATION:
A signal with continuous endpoints does not have spectral leakage. A signal with discontinuous endpoints has spectral leakage. A window can be used to restore continuity at the endpoints of a signal, but it can also introduce spectral leakage.
The amount of spectral leakage depends on the type of window used. A window with a sharp cutoff will introduce more spectral leakage than a window with a gradual cutoff.
OUTPUT:
(-1.0, 16.8)

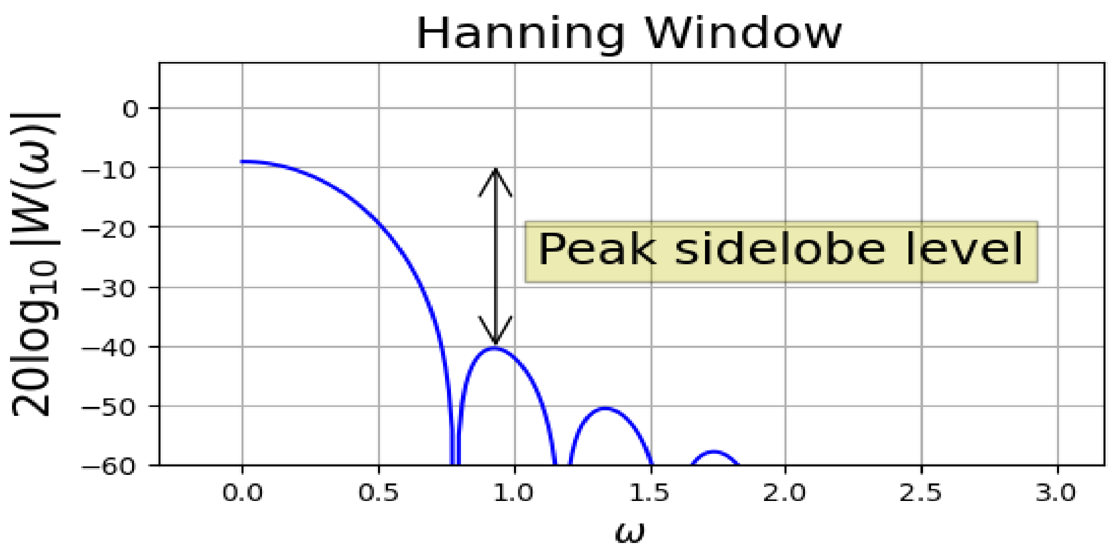


4.13
AIM:
To plots the magnitude of the Fourier transform of the Hanning window with the aim of the code is to illustrate the sidelobes of the Hanning window.

FUNCTION IMPORTED:

db20(): This function converts the magnitude of the Fourier transform to decibels.
signal.hann(): This function creates a Hanning window.
np.fft.fft(): This function computes the discrete Fourier transform of a signal.
np.arange(): This function generates an array of evenly spaced numbers.

OBSERVATION:
The observation from the code is that the Hanning window has sidelobes that are about 30 dB lower than the main lobe. The peak sidelobe level is located at the frequency that is 0.5 away from the main lobe.
The sidelobes of a window function are caused by the discontinuity at the endpoints of the window. The Hanning window has a smooth cutoff, which results in lower sidelobe levels than other windows with sharp cutoffs.
OUTPUT:
Text(0.4, 0.5, 'Peak sidelobe level')



4.14
AIM:
The aim of the code is to find the peak sidelobe level of a window function.

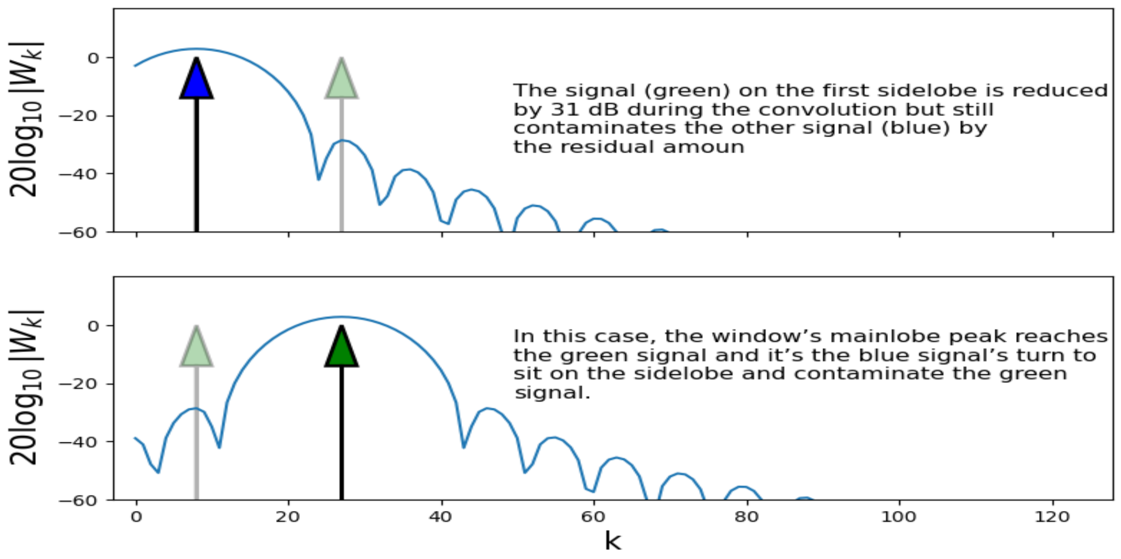
FUNCTION IMPORTED:
peak sidelobe(). It takes a window function as input and returns the peak sidelobe level and the bin index of the peak.
OBSERVATION:
The observation of the code is that the peak sidelobe level of a window function is related to the number of roots of the window function on the unit circle. The more roots the window function has on the unit circle, the lower the peak sidelobe level will be.

OUTPUT:
The output of the code will depend on the window function that is passed to the peak _sidelobe()

4.15
AIM:
The aim of the code appears to be the visualization and explanation of concepts related to sidelobes and signal contamination in the context of windowing and the Discrete Fourier Transform (DFT).It demonstrate how different signals interact with the sidelobes and mainlobes of a windowed spectrum.

FUNCTION IMPORTED:
The function used is db20(). It converts the magnitude of the DFT of a signal to decibels.
w = np.hanning(Ns) Creates a Hanning window of length Ns.
OBSERVATION:
The observation of the code is that the sidelobes of a window function can contaminate the signal being convolved with it. The more pronounced the sidelobes, the more the contamination.

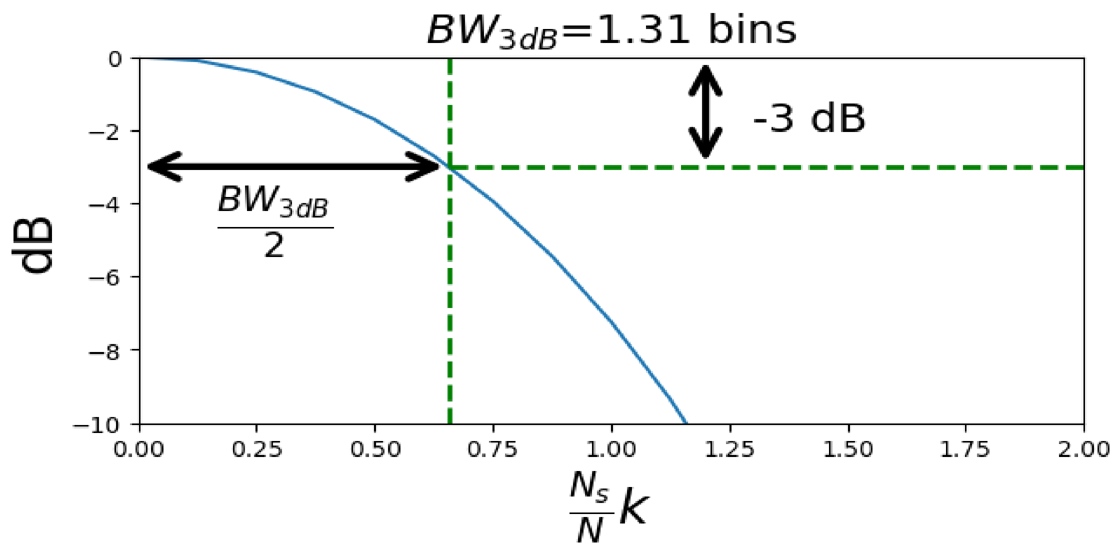
OUTPUT:



4.16
AIM:
The aim of the code is to show the 3-dB beamwidth of a Hamming window.

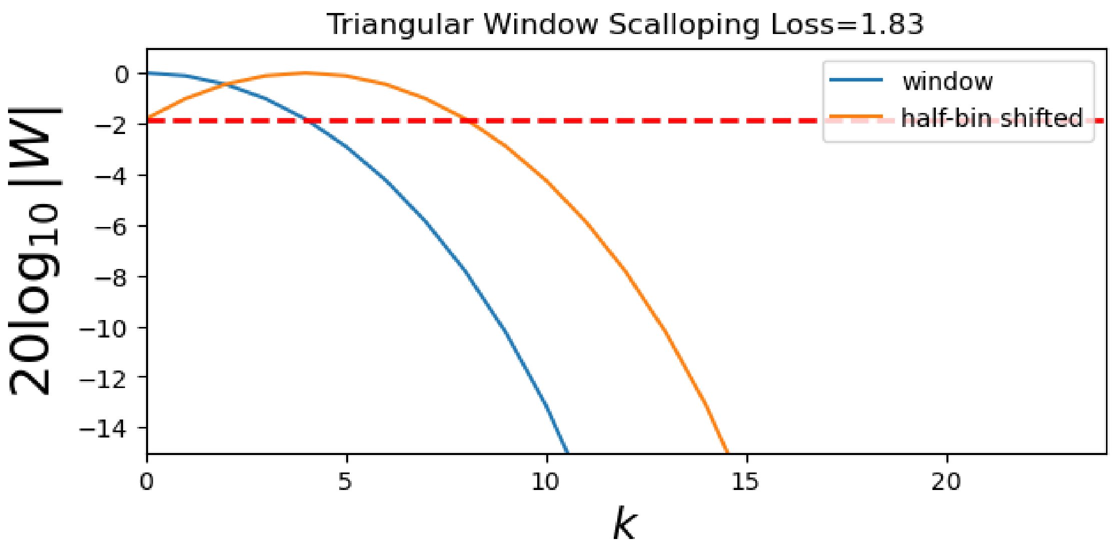
FUNCTION IMPORTED:
ax.vlines(x, ymin, ymax, ...):it is used to draw a dashed vertical line representing the 3-dB beamwidth.
ax.hlines(y, xmin, xmax, ...):it is used to draw a dashed horizontal line at -3 dB.
np.polyfit(x, y, degree):it is used to fit a quadratic polynomial to a set of points.

OBSERVATION:
The observation of the code is that the 3-dB beamwidth of a Hamming window is around 1.22 times the number of samples in the window.
In the code, the window function is first normalized to its peak value. Then, the DFT of the window function is calculated and the magnitude of the DFT is converted to decibels. A quadratic polynomial is fit to the first 10 decibels of the magnitude spectrum. The roots of the polynomial are the frequencies at which the magnitude spectrum drops by 3 dB. The width of the 3-dB beamwidth is twice the distance between the two roots. OUTPUT: Text(1.3, -2, '-3 dB')



4.17
AIM:
To illustrate the concept of scalloping loss associated with the triangular window function in signal processing
FUNCTION IMPORTED:
It uses the signal.windows.triang function from the signal module and the db20 function to analyze and visualize the behavior of the triangular window's frequency response.

OBSERVATION:
The observation of the code is that the scalloping loss of a triangular window is around -1.82 dB.
The plot should show how shifting the window by half a bin affects the frequency response and highlights the phenomenon of scalloping loss, where the response ”dips” between the original frequency bins. The legend differentiates between the window and the half-bin shifted response.
OUTPUT:


Triangular Window Scalloping Loss=1.83
window
half-bin shifted
$20\log_{10}|W|$
k
0
−2
−4
−6
−8
−10
−12
−14
0
5
10
15
20