Part1

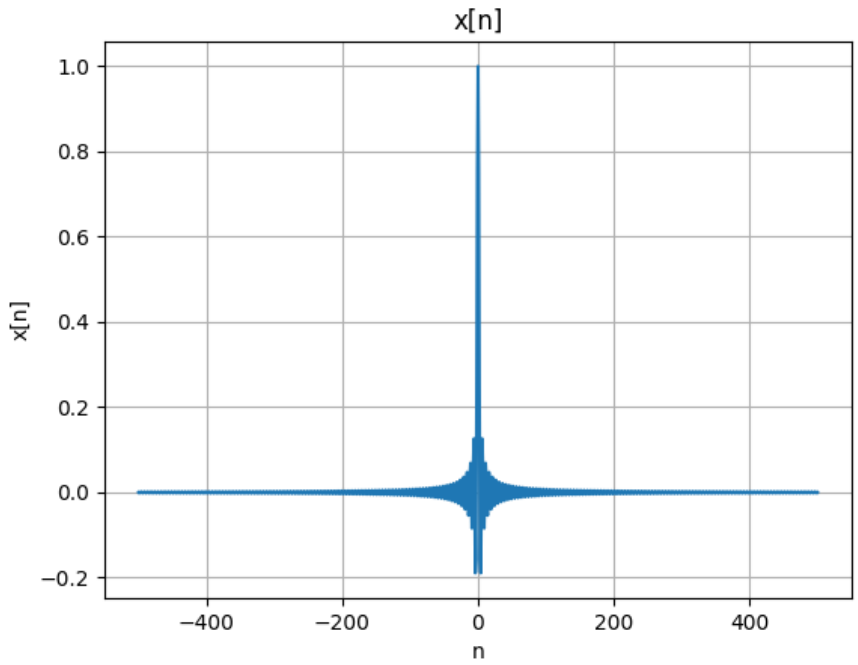
(a)

To get all the value of x[n] = x(n*T_s), we let n from -500 ~ 500 using np.linspace(). Then, we compute n*T_s for all n and store them in t. Finally, we compute the x(n*T_s) and get the result x[n]. Noted that the value of function at t = 0 is directly set to one since zero can not be divisor.

```
def fnc(t):
    y = np.ones_like(t)
    nonzero = t != 0
    y[nonzero] = np.sin(2 * np.pi * t[nonzero]) / (2 * np.pi * t[nonzero])
    return y

n = np.linspace(-500, 500, 1001)
t = n * (100/500)
x_n = fnc(t)
```

The result:

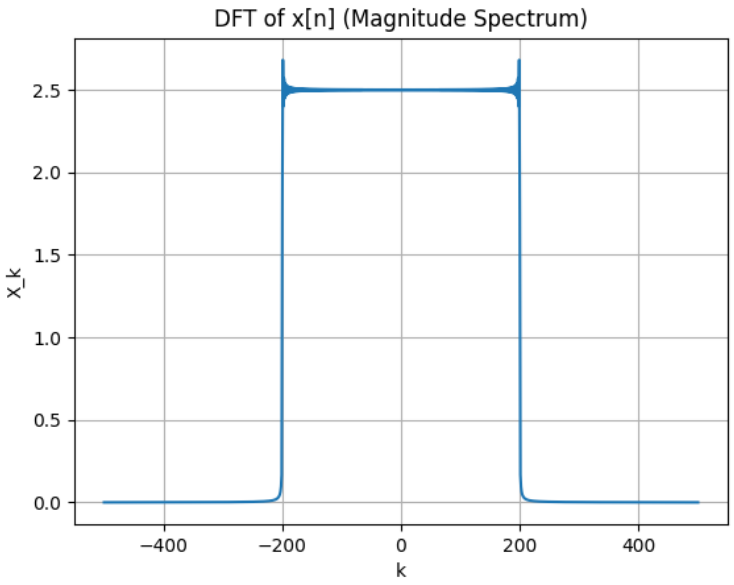


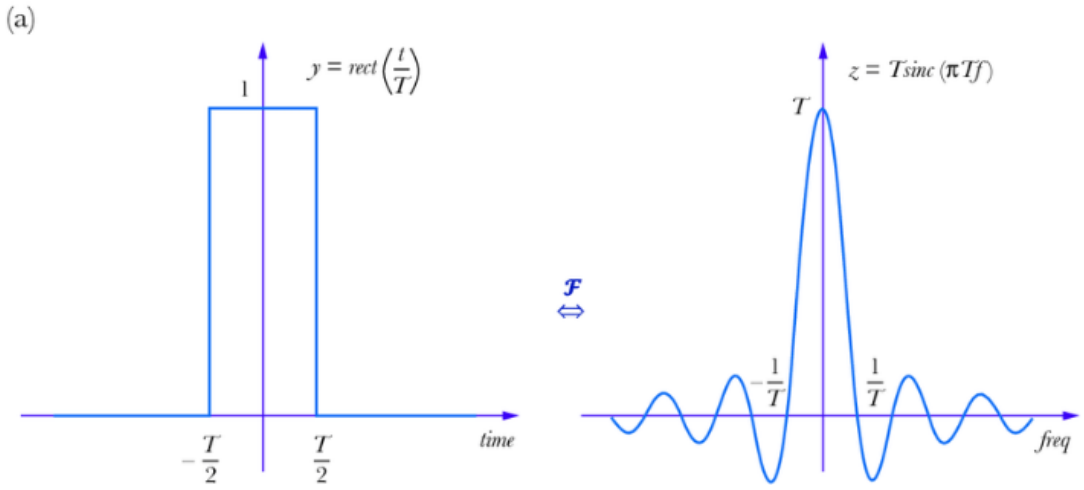
(b)

Compute DFT of x[n], and center the zero frequency using np.fft.fftshift(). Since we know that the Fourier transform of a discrete signal will be periodic with periodic 2*pi, we plot the result from -pi~pi, which is -500 * 2*pi/1001 ~ 500 * 2*pi/1001

```
x_k = np.fft.fft(x_n)
x_k_shifted = np.fft.fftshift(x_k) # 將低頻移到中心
freq = np.linspace(-500, 500, len(x_k_shifted))
```

The result:





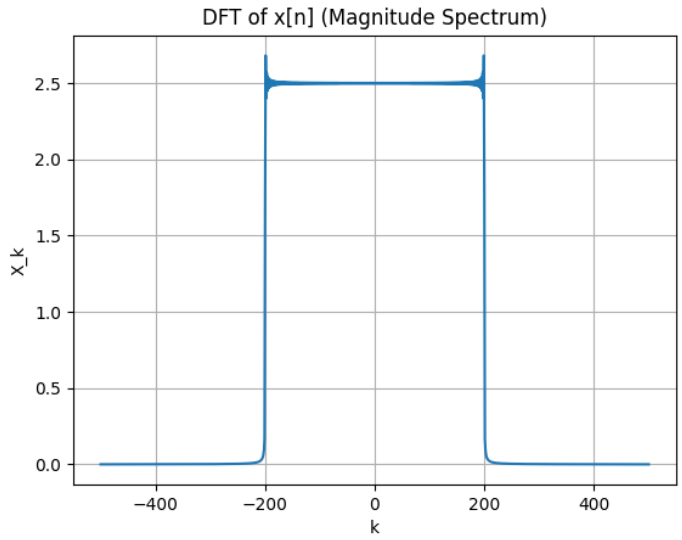
We can observe Gibbs phenomenon in this image. Since we know the Fourier transform of a Sinc function is a rectangular function, the Fourier transform of rectangular function will be Sinc function by duality. Now we only sample the value of Sinc function from t = -100 to 100 and discard all the high frequency parts. Therefore, if we implement Fourier transform to it, the low frequency parts cannot fully represent the impulse of rectangular function, i.e. the limited frequency components cannot perfectly reconstruct a discontinuous signal. We can observe some ripples there. These oscillations are known as the Gibbs phenomenon.

(c)

Compute the X_k for each k using Nested loop. We don't need to map n (-500 ~ 500) to (1~1001) to satisfy the equation (1). The result value will be the same since e^(-j2*pi/N * k) is equal to e^(-j2*pi/N * (k-1001)).

```
N = len(x_n)
X_manual = np.zeros(N, dtype=complex)
for k in range(N):
    for n in range(N):
        X_manual[k] += x_n[n] * np.exp(-1j * 2 * np.pi * k * n / N)
X_manual_shifted = np.fft.fftshift(X_manual)  # 將低頻移到中心
```

The result is the same as (b)



Part2

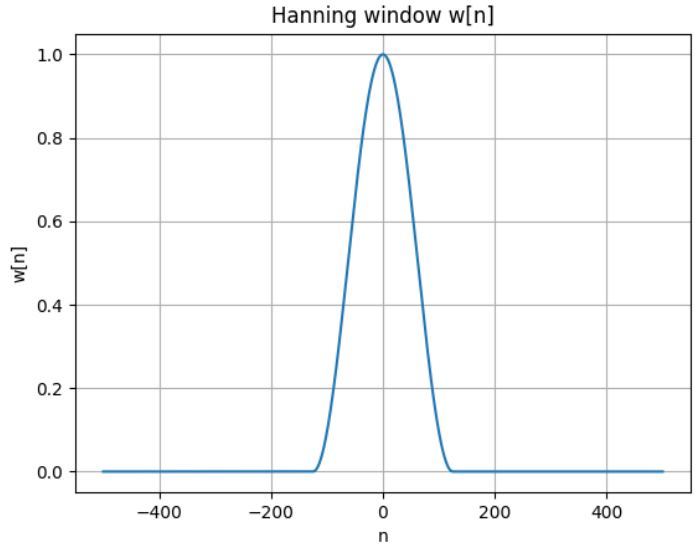
(a) We construct the Hanning window just like (a) in part one

```
def hanning_window(t, T0):
    w = np.zeros_like(t)
    inside = np.abs(t) <= (T0 / 2)  # Hanning window is non-zero only in this range
    w[inside] = 0.5 * (1 + np.cos(2 * np.pi * t[inside] / T0))
    return w

n = np.linspace(-500, 500, 1001)
t = n * (100/500)
T0 = 50

w_n = hanning_window(t, T0)
```

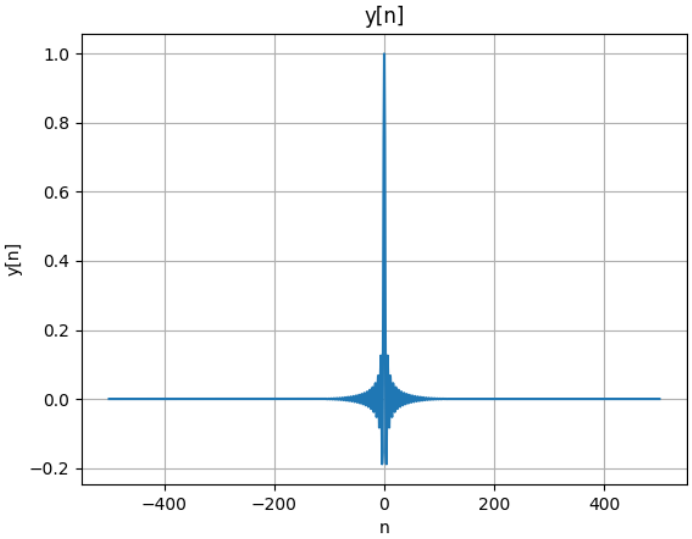
The result:



(b) We multiply x[n] and w[n] to get the y[n] and plot y[n] vs n

```
y_n = x_n * w_n
```

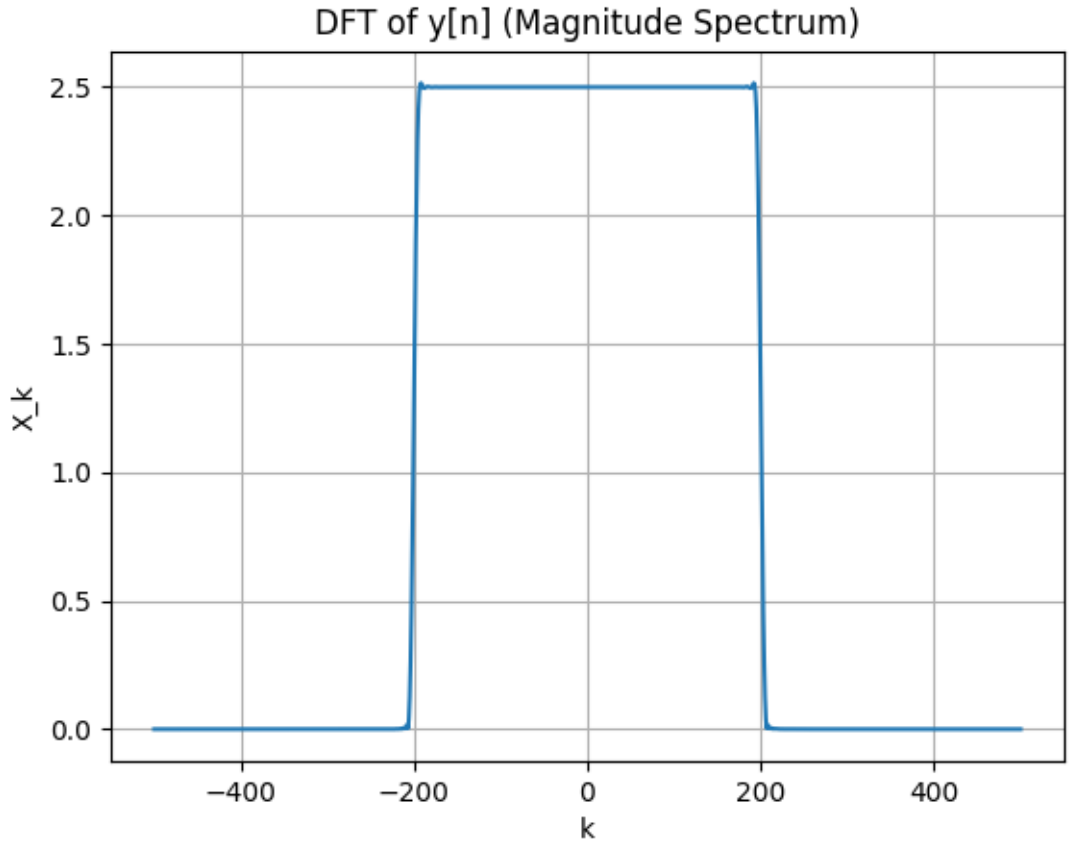
The result:



(c) We directly use numpy.fft.fft to compute DFT of y[n]

```
y_k = np.fft.fft(y_n)
y_k_shifted = np.fft.fftshift(y_k)
freq = np.linspace(-500, 500, len(y_k_shifted))
```

The result:



As we can see, the Gibbs phenomenon is mitigated. Hanning window reduces the influence of high frequency components which are the main cause of Gibbs oscillations. However, we can observe that the window also blurs sharp transition, the edge is not quite vertical and straight like before.