25 Spring ECEN 610: Mixed-Signal Interfaces

Lab2: Signal to Noise Ratio, Quantization

Name: Yu-Hao Chen

UIN:435009528

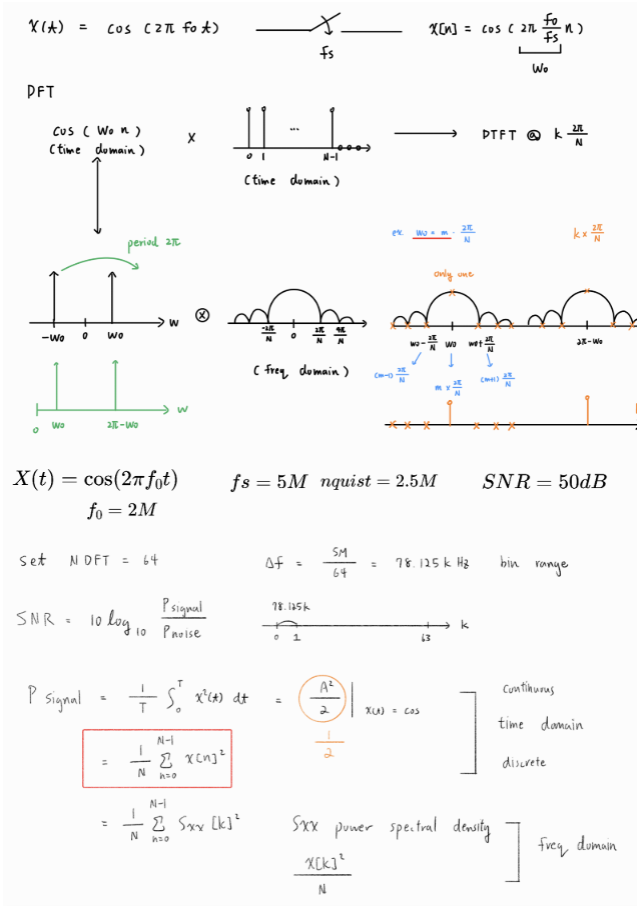
Section:601

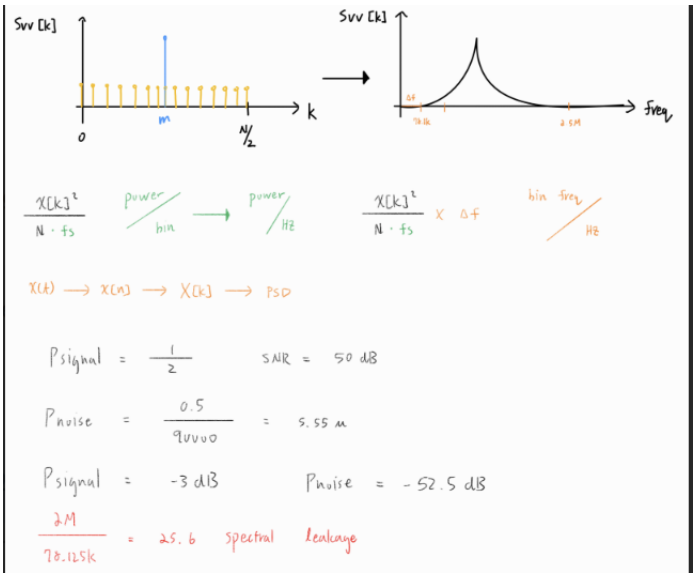
Professor: Sebastian Hoyos

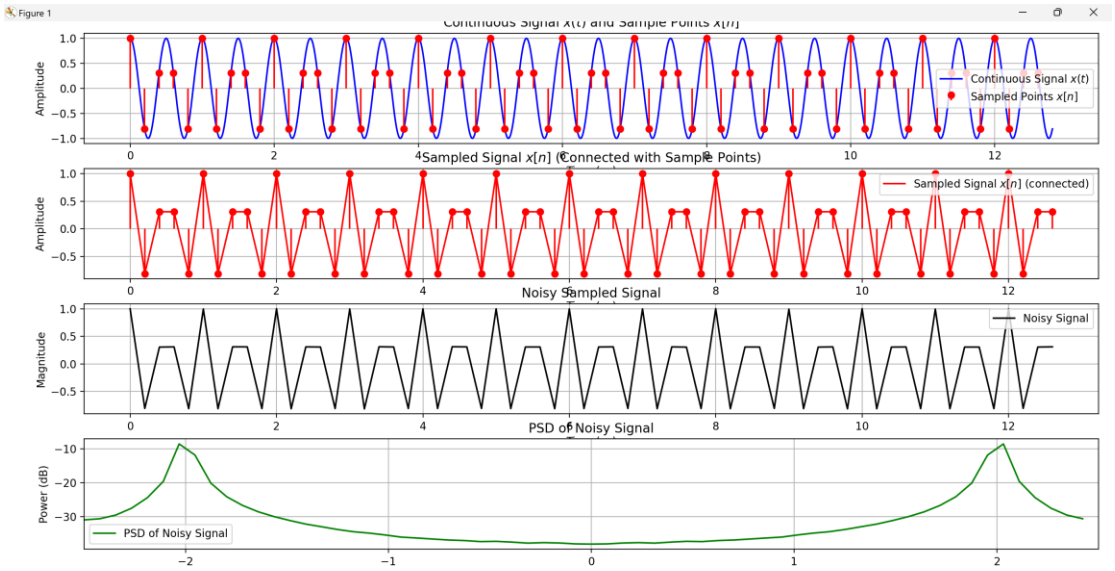
TA: Sky Zhao

1. SIGNAL TO NOISE RATIO (40%) Generate a tone with frequency 2 MHz and amplitude 1 V. Sample the tone at frequencies Fs = 5 MHz.

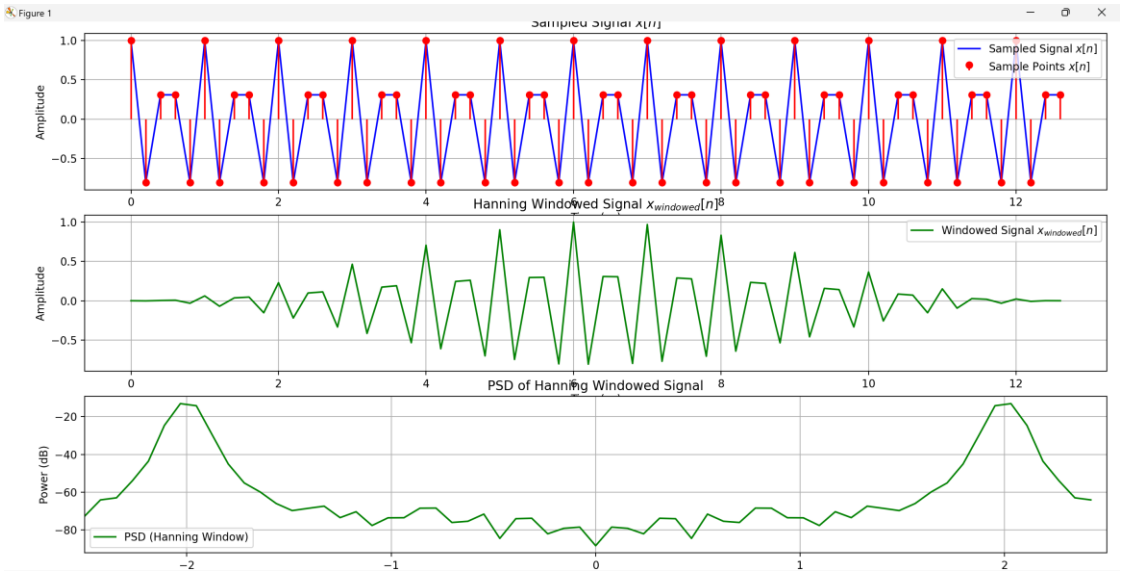
a) Add Gaussian noise to the sampled sinewave such that the signal SNR is 50 dB. Find first the variance of the Gaussian noise needed to produce the target SNR. Calculate and plot the Power Spectral Density (PSD) from the DFT of the noisy samples. Corroborate that the SNR calculation from the DFT plot gives the theoretical result. What would be the variance of a uniformly distributed noise to obtain the same SNR.



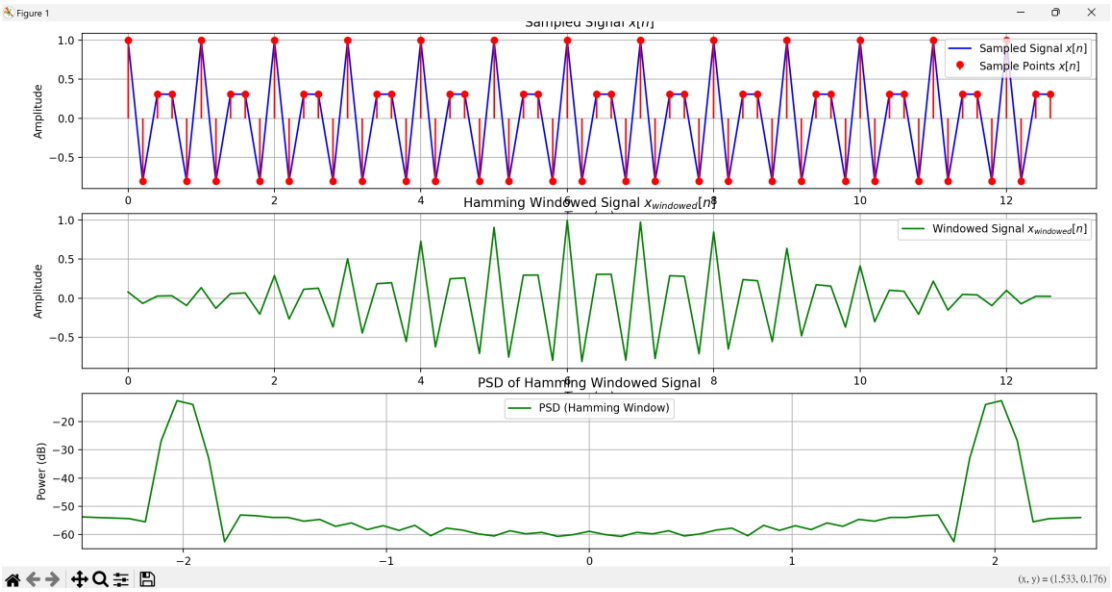




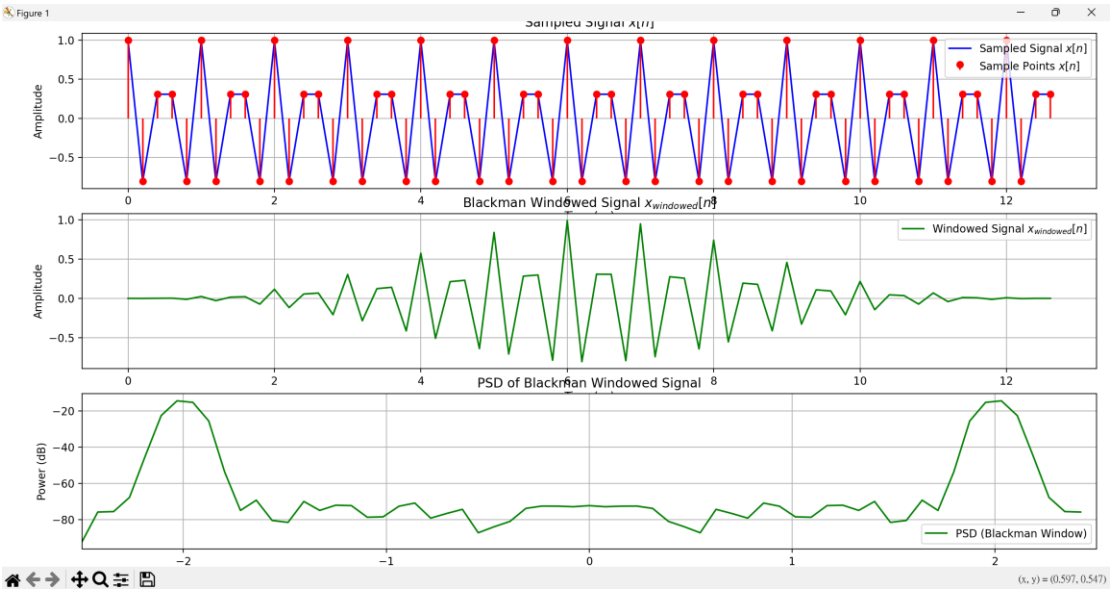
b) Now repeat a.) using a window before the DFT. Use the following windows: Hanning, Hamming, Blackman. What are your conclusions? NOTE: The use of windows mentioned above spreads the signal power. You must take this into account when computing SNR.



Hanning window



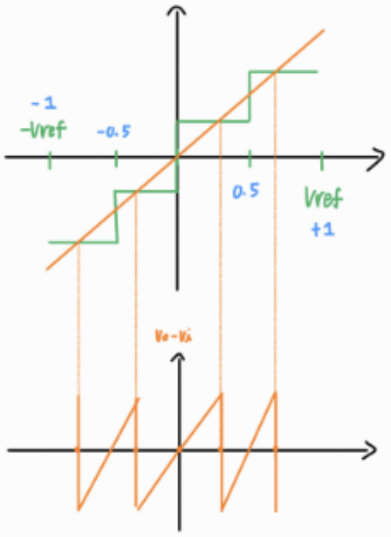
Hamming window



Blackman window

2. QUANTIZATION

a) Create a perfect quantizer with 6 bits of resolution and with flexible sampling rate. For a 200 MHz full scale input tone, sample and quantize the sinewave at 400 MHz and plot the PSD of 30 periods. What is the SNR? Repeat the SNR calculation for 100 periods of the same signal. Make your own conclusions about this test regarding periodicity of quantization noise and the impact of this in the SNR. How can you solve this problem?



2bit $2^2 = 4$ level

$$\Delta V = \frac{2Vref}{2^n} = \frac{2}{4} = 0.5$$

$$\overline{q^2} = \frac{\Delta^2}{12}$$

6 bits : $2^6 = 64$ $\Delta = \frac{2 \times Vref}{2^6} = 0.03125$

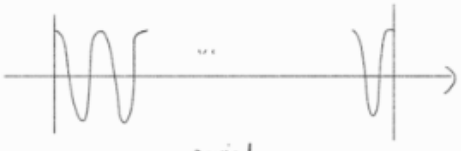
$$P_{signal} = \frac{A^2}{2} = 1$$

$$q\ noise = \frac{\Delta^2}{12} = 8.14 \times 10^{-5}$$

$$SNR\ dB = 6.02 \times 6 + 1.76 = 38\ dB$$

$f_0 = 400M\ Hz$ $f_s \geq 2 \times 400M$ set $f_s = 2G\ Hz$

$X(t) = \cos(2\pi \cdot 400\ M\ Hz \cdot t)$

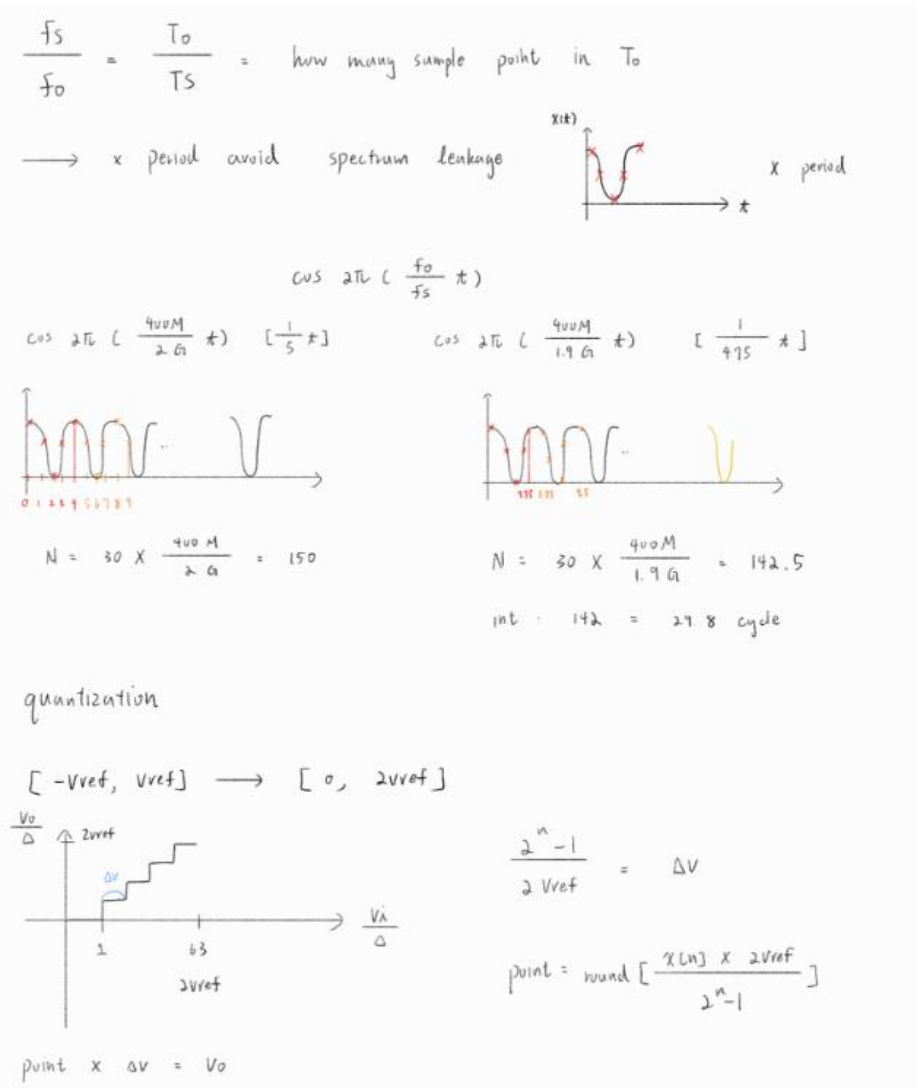


$$\Delta f = \frac{f_s}{N} \qquad N = period \times \frac{f_s}{f_0}$$

$$\frac{2G}{400M} = 5$$

period = 30 $N = 150$ $\Delta f = \frac{2G}{150} = 13.33\ M\ Hz$

period = 100 $N = 500$ $\Delta f = \frac{2G}{500} = 4\ M\ Hz$

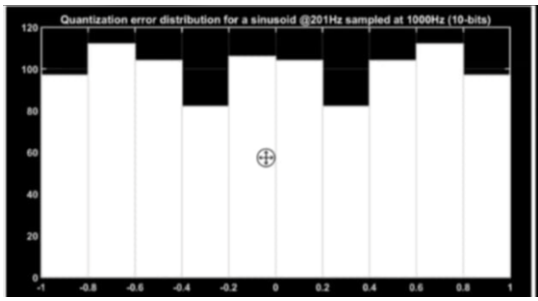


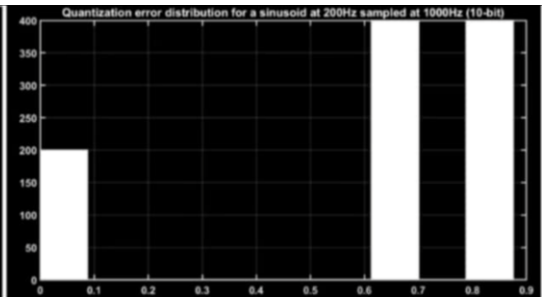
Ex: Fs= 2G, F0= 400M, cycle=30, Fs/F0=5

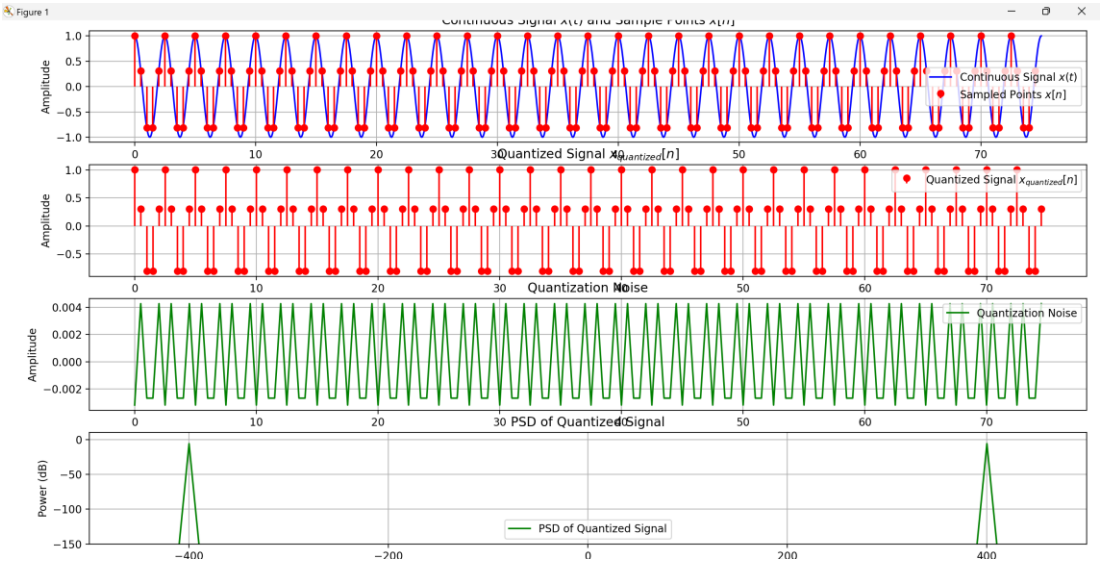
N=30*5=150, 150/5= 30cycle (periodic)

If sampling frequency Fs is **A times** the input signal frequency F0, this creates **periodic quantization error** because the sampling points repeat every Fs/F0 cycles. Periodic error accumulates at certain frequencies, causing **lower noise power** and an **artificially high SNR**. (quantization error distribution is not like a white noise)

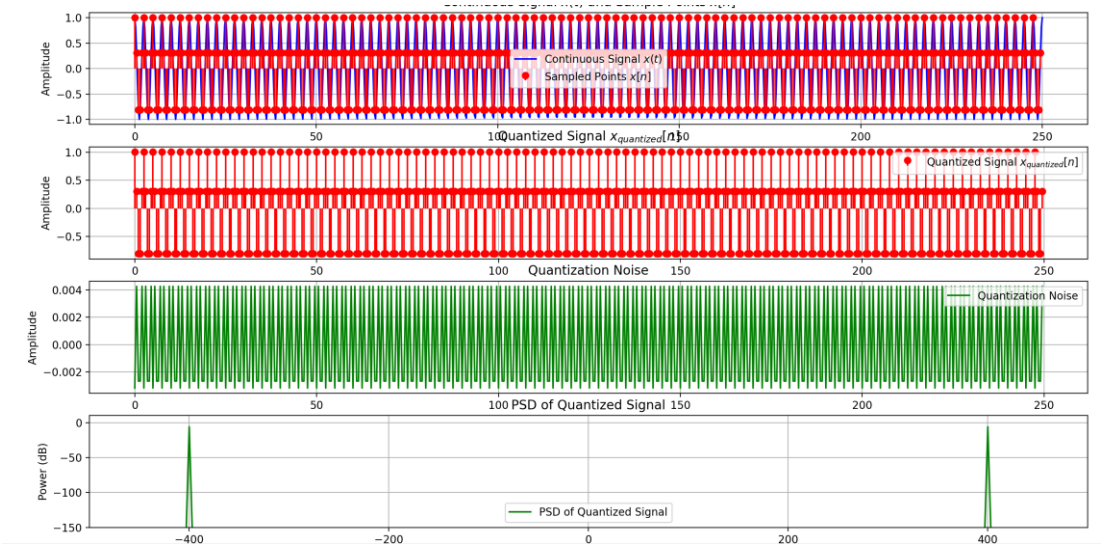
Change NNN-DFT, period, or Fs to avoid sampling period being an integer multiple of the signal period.








```
=== SNR for 30 Cycles ===
SNR = 43.53 dB
```



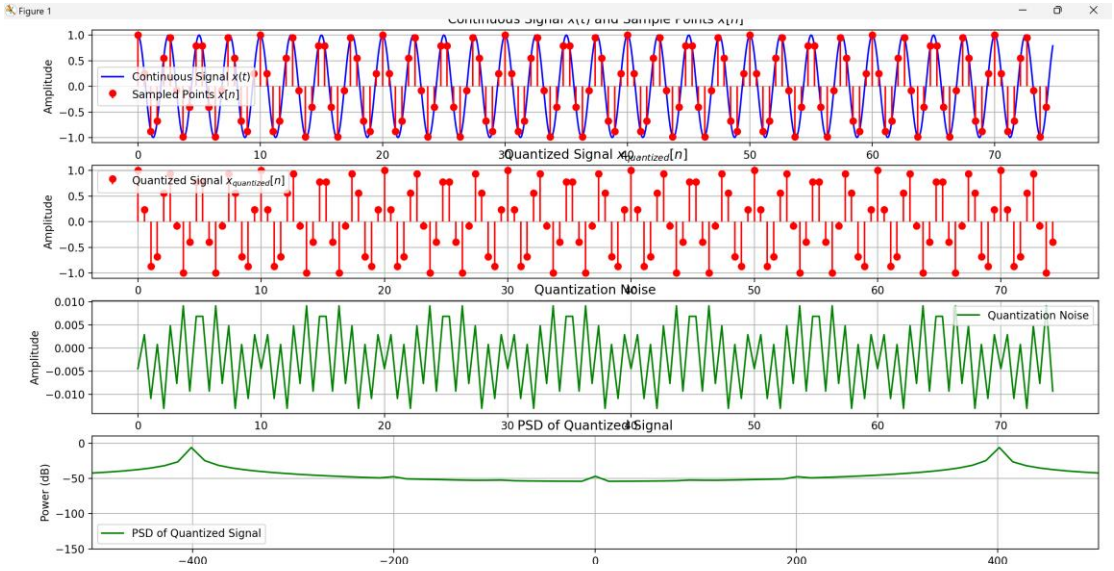

```
=== SNR for 100 Cycles ===
SNR = 43.53 dB
```

Increasing cycle will make the spectrum clearer (increase N) and more stable, but the amount of calculation will increase. The quantization error repeats periodically: it will cause harmonic distortion (Harmonic Distortion) and make the SNR higher.

b) Find an incommensurate sampling frequency larger than Nyquist rate. Plot the PSD of the new samples. Calculate the SNR from the figure.

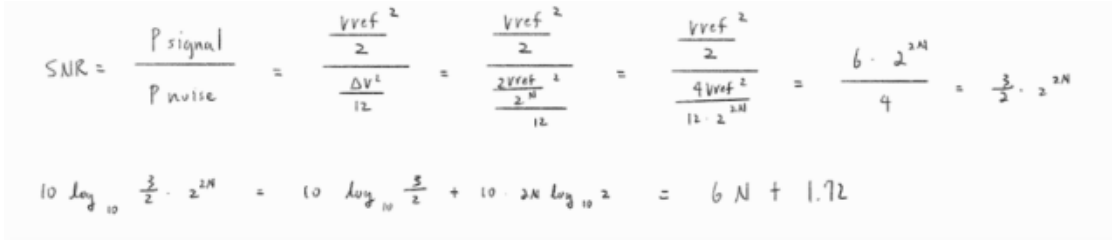
Ex: Fs= 1.9G, F0= 400M, Fs/F0=4.75,

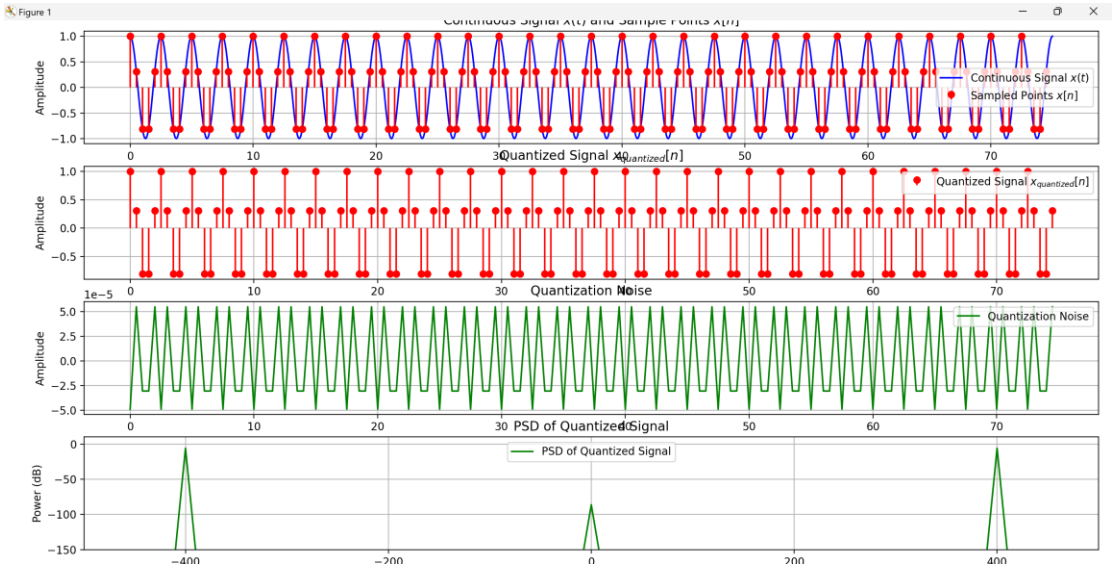
N= cycle*4.75= 142.5, DFT int(N)= 142/4.75= 29.89 cycle, Spectral Leakage



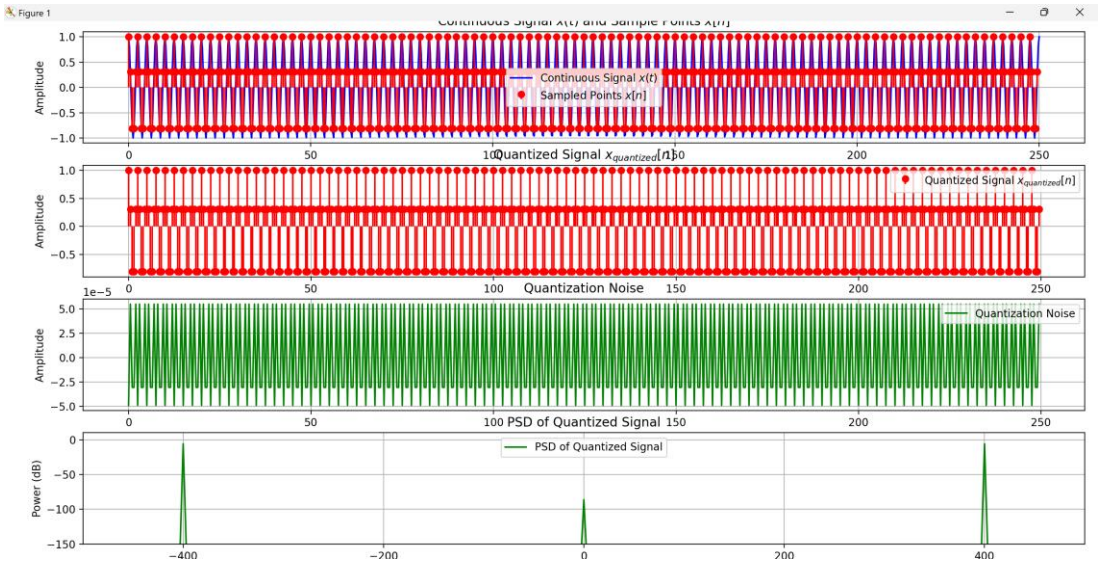

```
=== SNR for 29.8 Cycles ===
SNR = 38.80 dB
PS C:\Users\harri\OneDrive\桌面\TAMU\ECEN 610 Mixed-signal\lab2 quantization+noise>
```

c) Repeat a) using a 12 bit quantizer. Can you prove from simulations that SNR ~ 6N (where N is the number of bits used by the quantizer) in both the cases, N = 6 and N = 12?



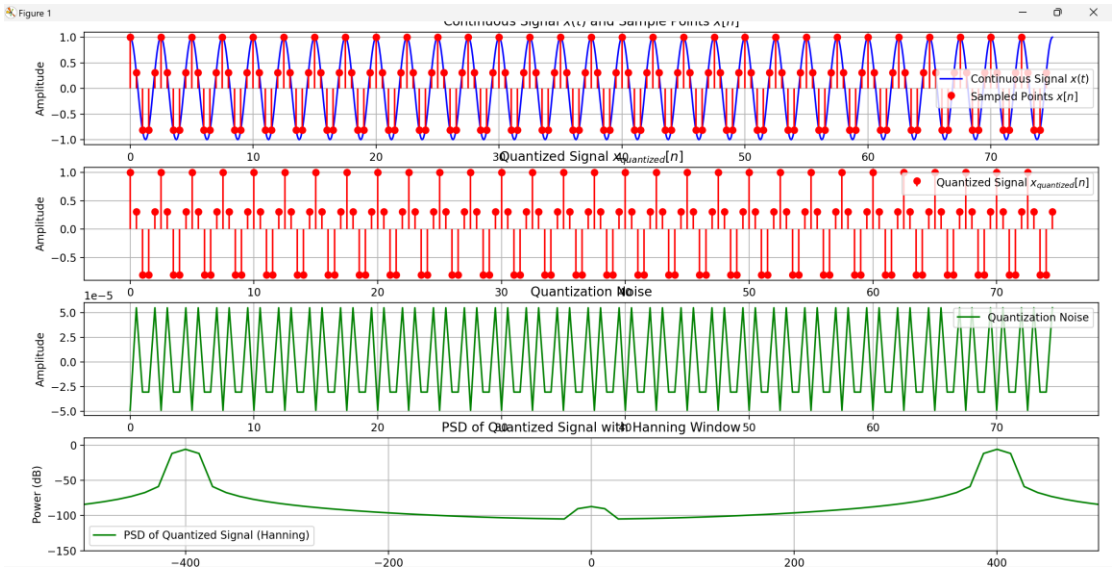



```
=== SNR for 30 Cycles ===
SNR = 80.52 dB
PS C:\Users\harri\OneDrive\桌面\TAMU\ECEN 610 Mixed-signal\lab2 quantization+noise>
```

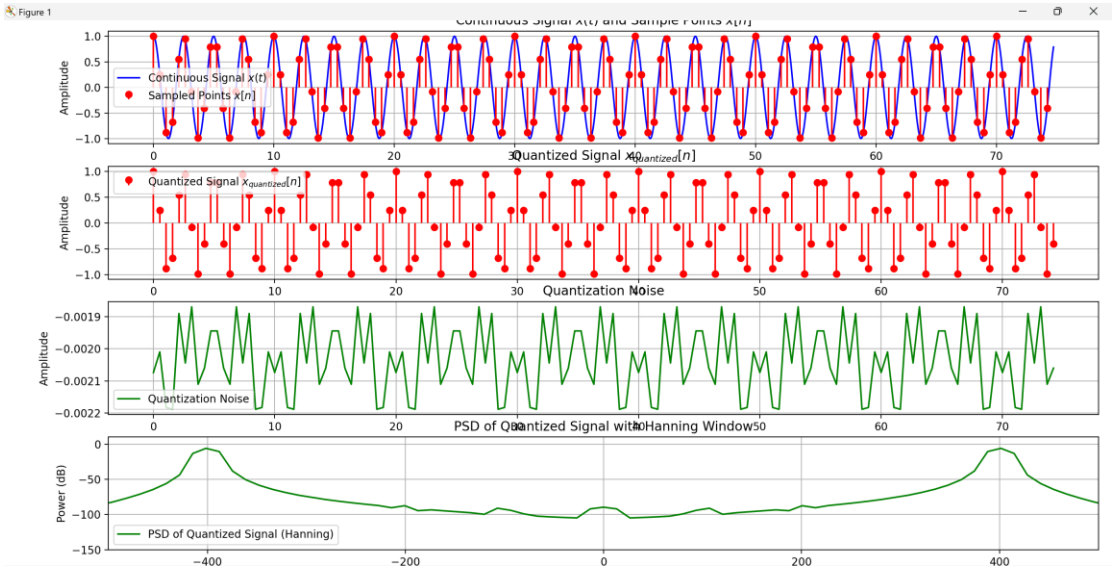



```
=== SNR for 100 Cycles ===
SNR = 80.52 dB
PS C:\Users\harri\OneDrive\桌面\TAMU\ECEN 610 Mixed-signal\lab2 quantization+noise>
```

d) Use a Hanning window and repeat c). What is the SNR? Make your own conclusions.

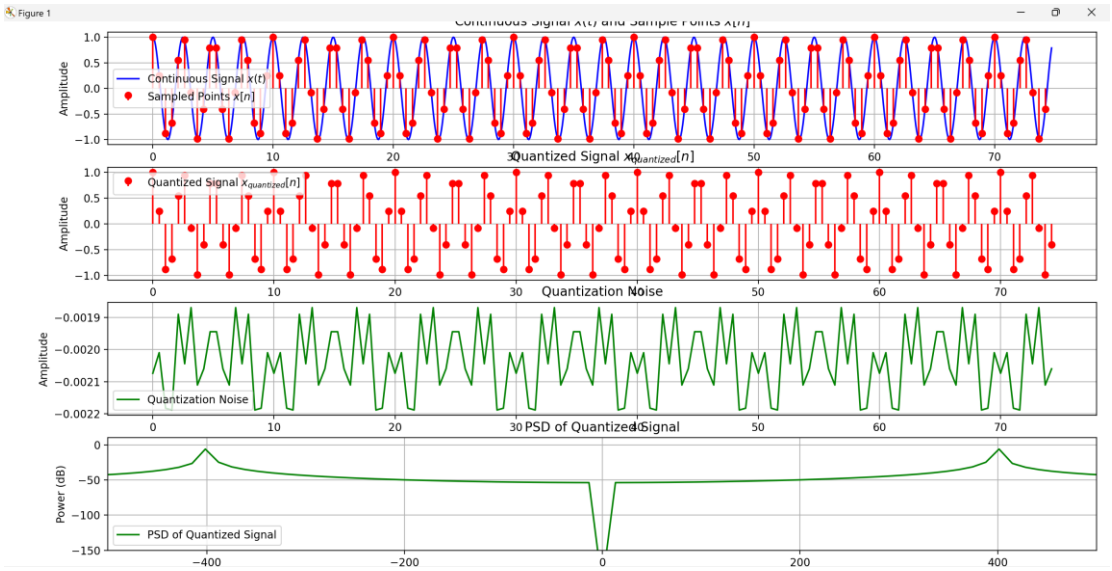



```
=== SNR for 30 Cycles with Hanning Window ===
SNR = 80.52 dB
```




```
=== SNR for 30 Cycles with Hanning Window ===
SNR = 75.87 dB
```

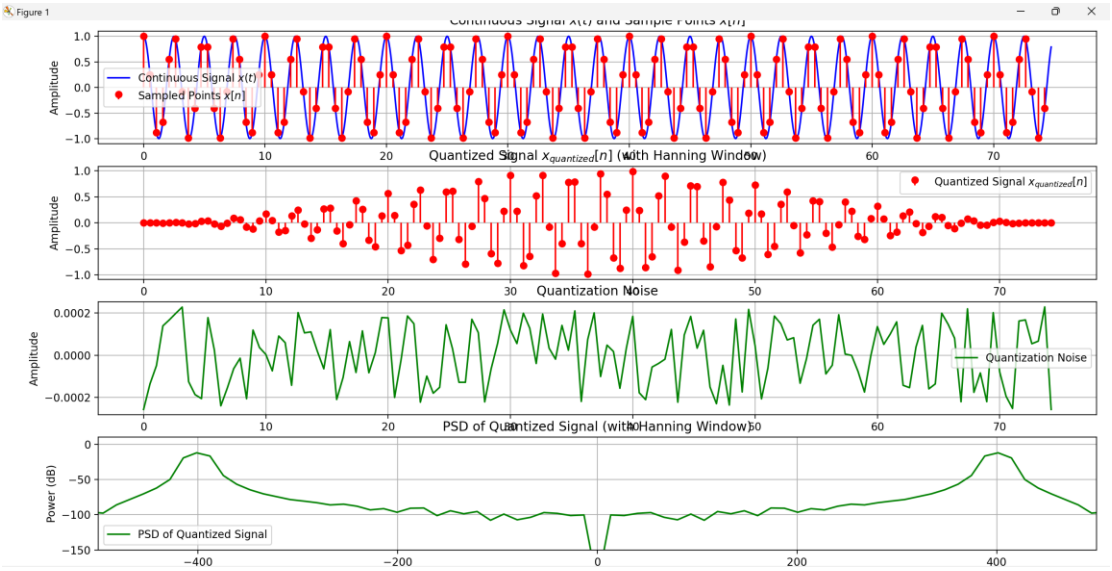
e) Now add noise again so the signal SNR is 38 dB . Repeat c) and d). What are the SNRs? Provide conclusions.




```
=== SNR for 30 Cycles (No Window) ===
SNR = 75.87 dB
```

```
Frequency: 294.37 MHz, PSD: -44.18 dB
Frequency: 307.75 MHz, PSD: -42.96 dB
Frequency: 321.13 MHz, PSD: -41.54 dB
Frequency: 334.51 MHz, PSD: -39.87 dB
Frequency: 347.89 MHz, PSD: -37.83 dB
Frequency: 361.27 MHz, PSD: -35.18 dB
Frequency: 374.65 MHz, PSD: -31.43 dB
Frequency: 388.03 MHz, PSD: -24.84 dB
Frequency: 401.41 MHz, PSD: -6.17 dB
Frequency: 414.79 MHz, PSD: -26.51 dB
Frequency: 428.17 MHz, PSD: -32.03 dB
Frequency: 441.55 MHz, PSD: -35.31 dB
Frequency: 454.93 MHz, PSD: -37.65 dB
Frequency: 468.31 MHz, PSD: -39.45 dB
Frequency: 481.69 MHz, PSD: -40.90 dB
Frequency: 495.07 MHz, PSD: -42.12 dB
Frequency: 508.45 MHz, PSD: -43.16 dB
```

```
Frequency: -495.07 MHz, PSD: -42.12 dB
Frequency: -481.69 MHz, PSD: -40.90 dB
Frequency: -468.31 MHz, PSD: -39.45 dB
Frequency: -454.93 MHz, PSD: -37.65 dB
Frequency: -441.55 MHz, PSD: -35.31 dB
Frequency: -428.17 MHz, PSD: -32.03 dB
Frequency: -414.79 MHz, PSD: -26.51 dB
Frequency: -401.41 MHz, PSD: -6.17 dB
Frequency: -388.03 MHz, PSD: -24.84 dB
Frequency: -374.65 MHz, PSD: -31.43 dB
Frequency: -361.27 MHz, PSD: -35.18 dB
Frequency: -347.89 MHz, PSD: -37.83 dB
Frequency: -334.51 MHz, PSD: -39.87 dB
Frequency: -321.13 MHz, PSD: -41.54 dB
```




```
=== SNR for 30 Cycles (Hanning Window) ===
SNR = 69.41 dB
```

```
Frequency: 334.51 MHz, PSD: -70.41 dB
Frequency: 347.89 MHz, PSD: -64.66 dB
Frequency: 361.27 MHz, PSD: -56.77 dB
Frequency: 374.65 MHz, PSD: -44.42 dB
Frequency: 388.03 MHz, PSD: -16.78 dB
Frequency: 401.41 MHz, PSD: -12.16 dB
Frequency: 414.79 MHz, PSD: -19.48 dB
Frequency: 428.17 MHz, PSD: -50.07 dB
Frequency: 441.55 MHz, PSD: -62.14 dB
Frequency: 454.93 MHz, PSD: -70.58 dB
Frequency: 468.31 MHz, PSD: -78.31 dB
Frequency: 481.69 MHz, PSD: -86.05 dB
Frequency: 495.07 MHz, PSD: -97.75 dB
```

```
Frequency: -481.69 MHz, PSD: -86.05 dB
Frequency: -468.31 MHz, PSD: -78.31 dB
Frequency: -454.93 MHz, PSD: -70.58 dB
Frequency: -441.55 MHz, PSD: -62.14 dB
Frequency: -428.17 MHz, PSD: -50.07 dB
Frequency: -414.79 MHz, PSD: -19.48 dB
Frequency: -401.41 MHz, PSD: -12.16 dB
Frequency: -388.03 MHz, PSD: -16.78 dB
Frequency: -374.65 MHz, PSD: -44.42 dB
Frequency: -361.27 MHz, PSD: -56.77 dB
Frequency: -347.89 MHz, PSD: -64.66 dB
Frequency: -334.51 MHz, PSD: -70.41 dB
Frequency: -321.13 MHz, PSD: -74.44 dB
Frequency: -307.75 MHz, PSD: -78.50 dB
```

The window will have minor effect on intergrate cycle casue there are no specturm leakage for intergrate cycle

- Appendix

Q1.a PSD noise (without quantization)

```python
# === continuous time signal ===
t_cont = np.linspace(0, N * Ts, 1000)  # Continuous time axis (1000 points)
x_t = np.cos(2 * np.pi * F * t_cont)   # Continuous time signal x(t)

# === discrete time signal ===
t_n = np.arange(N) * Ts  # Discrete time axis: 64 points
x_n = np.cos(2 * np.pi * F * t_n)  # Sampled signal x[n]

# === POWER signal & noise ===
signal_power = np.mean(x_n ** 2)  # P signal= 1/N (∑ (n=0~N-1) x[n]²), x[n] time domain
noise_power = signal_power / (10 ** (SNR_dB / 10))  # SNR= 10 log10(P signal/ P noise)
noise = np.sqrt(noise_power)  # avg= 0 white noise, P noise= σ² noise

# === Gaussian Noise ===
Gaussian_noise = np.random.normal(0, noise, N)  # Gaussian noise
x_noisy = x_n + Gaussian_noise  # Noisy signal

# === FFT ===
X_k = np.fft.fft(x_noisy, N)  # DFT freq amp x[k]
X_k_shifted = np.fft.fftshift(X_k)  # Move center to 0
frequencies = np.fft.fftfreq(N, Ts)  # DFT freq (0, 78.125k, ..., 2.5M)
frequencies_shifted = np.fft.fftshift(frequencies)  # Move center to 0

# === PSD  ===
df = Fs / N # bin value: 78.125k
PSD = (np.abs(X_k_shifted) ** 2 / (N * Fs)) * df  # [PSD(bin)/fs= power/hz] * bin's Hz (df)= PSD power/bin
PSD_dB = 10 * np.log10(PSD)  # dB
```

Hanning window

```python
Fs = 5e6
F = 2e6
N = 64
Ts = 1 / Fs
SNR_dB = 50

# === 1.(Sampled Signal) ===
t_n = np.arange(N) * Ts  # discrete time axis
x_n = np.cos(2 * np.pi * F * t_n)  # sample signal

# === 2.Gaussian noise ===
signal_power = np.mean(x_n ** 2)   # signal power
noise_power = signal_power / (10 ** (SNR_dB / 10))  # noise power
noise_std = np.sqrt(noise_power)   # noise standard
Gaussian_noise = np.random.normal(0, noise_std, N)  # Gaussian Noise
x_noisy = x_n + Gaussian_noise      # signal with noise

# === 3. Hanning Window ===
hanning_window = np.hanning(N)  # Hanning Window
x_windowed = x_noisy * hanning_window  # signal with noise + Window

# === 4. FFT & freq axis ===
X_k = np.fft.fft(x_windowed, N)  # FFT Spectrum
X_k_shifted = np.fft.fftshift(X_k)  # center 0
frequencies = np.fft.fftfreq(N, Ts)  # FFT freq axis (0, 78.125k, ... 2.5M)
frequencies_shifted = np.fft.fftshift(frequencies)  # center 0

# === 5. Power Spectral Density (PSD) ===
df = Fs / N
PSD = (np.abs(X_k_shifted) ** 2 / (N * Fs)) * df  # PSD
PSD_dB = 10 * np.log10(PSD)  # dB 單位
```

Hamming window

```python
# === 3. Hamming Window ===
hanning_window = np.hamming(N)  # Hanning Window
x_windowed = x_noisy * hanning_window  # signal with noise + Window
```

Blackman window

```python
# === 3. Blackman Window ===
hanning_window = np.blackman(N)  # Hanning Window
x_windowed = x_noisy * hanning_window  # signal with noise + Window
```

Q2 N cycle

```python
Fs = 2e9
F = 4e8
N_bits = 6
Vref = 1
quantization_levels = 2 ** N_bits

def simulate_quantization(N_cycles):
    # === 1. x[n] ===
    N = int(N_cycles * Fs / F)  # N= int (period) * (T0/Ts= sample point in one T0)
    t_n = np.arange(N) / Fs     # x[n] tiem axis: n * Ts, n= 0, 1, 2, 3... N-1
    t_cont = np.linspace(0, N / Fs, 1000)  # x(t) time axis
    x_input_cont = Vref * np.cos(2 * np.pi * F * t_cont)  # x(t)
    x_input = Vref * np.cos(2 * np.pi * F * t_n)  # x[n]

    # === 2. Quantization ===
    x_quantized = np.round((x_input + Vref) * (quantization_levels - 1) / (2 * Vref)) * (2 * Vref) / (quantization_levels - 1) - Vref
    x_quantized_nodc = x_quantized - np.mean(x_quantized)
    quantization_noise = x_input - x_quantized  # Quantization error
    quantization_noise_nodc = x_input - x_quantized_nodc

    # === 3. SNR ===
    signal_power = np.mean(x_input ** 2)  # P signal
    noise_power = np.mean(quantization_noise ** 2)
    SNR = 10 * np.log10(signal_power / noise_power)

    # === 5. FFT & PSD ===
    X_k = np.fft.fft(x_quantized, N)
    X_k_shifted = np.fft.fftshift(X_k)
    frequencies = np.fft.fftfreq(N, 1 / Fs)
    frequencies_shifted = np.fft.fftshift(frequencies)
    df = Fs / N
    PSD = (np.abs(X_k_shifted) ** 2 / (N * Fs)) * df
    PSD_dB = 10 * np.log10(PSD + 1e-20)
```

Q2.b not int cycle

```python
Fs = 1.9e9
F = 4e8
N_bits = 6
Vref = 1
quantization_levels = 2 ** N_bits

def simulate_quantization(N_cycles):
    # === 1. x[n] ===
    N = int(N_cycles * Fs / F)  # N= int (period= 30) * (T0/Ts= 4.75)= 142.5, 142 (DFT only int)= 29.89cycle
    t_n = np.arange(N) / Fs     # x[n] tiem axis: n * Ts, n= 0, 1, 2, 3... N-1
    t_cont = np.linspace(0, N / Fs, 1000)  # x(t) time axis
    x_input_cont = Vref * np.cos(2 * np.pi * F * t_cont)  # x(t)
    x_input = Vref * np.cos(2 * np.pi * F * t_n)  # x[n]

    # === 2. Quantization ===
    x_quantized = np.round((x_input + Vref) * (quantization_levels - 1) / (2 * Vref)) * (2 * Vref) / (quantization_levels - 1) - Vref
    x_quantized_nodc = x_quantized - np.mean(x_quantized)
    quantization_noise = x_input - x_quantized  # Quantization error
    quantization_noise_nodc = x_input - x_quantized_nodc

    # === 3. SNR ===
    signal_power = np.mean(x_input ** 2)  # P signal
    noise_power = np.mean(quantization_noise ** 2)
    SNR = 10 * np.log10(signal_power / noise_power)

    # === 5. FFT & PSD ===
    X_k = np.fft.fft(x_quantized, N)
    X_k_shifted = np.fft.fftshift(X_k)
    frequencies = np.fft.fftfreq(N, 1 / Fs)
    frequencies_shifted = np.fft.fftshift(frequencies)
    df = Fs / N
    PSD = (np.abs(X_k_shifted) ** 2 / (N * Fs)) * df
    PSD_dB = 10 * np.log10(PSD + 1e-20)
```

Q2 with window

```python
Fs = 1.9e9
F = 4e8
N_bits = 12
Vref = 1
quantization_levels = 2 ** N_bits

def simulate_quantization(N_cycles):
    # === 1. x[n] ===
    N = int(N_cycles * Fs / F)  # N= int (period) * (T0/Ts= sample point in one T0)
    t_n = np.arange(N) / Fs     # x[n] time axis: n * Ts, n= 0, 1, 2, 3... N-1
    t_cont = np.linspace(0, N / Fs, 1000)  # x(t) time axis
    x_input_cont = Vref * np.cos(2 * np.pi * F * t_cont)  # x(t)
    x_input = Vref * np.cos(2 * np.pi * F * t_n)  # x[n]

    # === 2. Quantization ===
    x_quantized = np.round((x_input + Vref) * (quantization_levels - 1) / (2 * Vref)) * (2 * Vref) / (quantization_levels
    x_quantized_nodc = x_quantized - np.mean(x_quantized)
    quantization_noise = x_input - x_quantized  # Quantization error
    quantization_noise_nodc = x_input - x_quantized_nodc

    # === 3. Hanning Window ===
    hanning_window = np.hanning(N)
    x_windowed = x_quantized * hanning_window
    correction_factor = np.sum(hanning_window) / N  # Window Correction Factor

    # === 4. SNR ===
    signal_power = np.mean(x_input ** 2)  # P signal
    noise_power = np.mean(quantization_noise ** 2)
    SNR = 10 * np.log10(signal_power / noise_power)

    # === 5. FFT & PSD ===
    X_k = np.fft.fft(x_windowed, N) / correction_factor  # Apply Hanning Window
```

Q2 noise window

```python
Fs = 1.9e9
F = 4e8
N_bits = 12
Vref = 1
quantization_levels = 2 ** N_bits

def simulate_quantization(N_cycles, use_window=False):
    # === 1. x[n] ===
    N = int(N_cycles * Fs / F)  # N= int (period) * (T0/Ts= sample point in one T0)
    t_n = np.arange(N) / Fs     # x[n] time axis: n * Ts, n= 0, 1, 2, 3... N-1
    t_cont = np.linspace(0, N / Fs, 1000)  # x(t) time axis
    x_input_cont = Vref * np.cos(2 * np.pi * F * t_cont)  # x(t)
    x_input = Vref * np.cos(2 * np.pi * F * t_n)  # x[n]

    # === 2. Apply Hanning Window (Optional) ===
    if use_window:
        window = np.hanning(N)
        x_windowed = x_input * window
    else:
        x_windowed = x_input

    # === 3. Quantization ===
    x_quantized = np.round((x_windowed + Vref) * (quantization_levels - 1) / (2 * Vref)) * (2 * Vref) / (quantization_lev
    x_quantized_nodc = x_quantized - np.mean(x_quantized)
    quantization_noise = x_windowed - x_quantized  # Quantization error
    quantization_noise_nodc = x_windowed - x_quantized_nodc

    # === 4. SNR ===
    signal_power = np.mean(x_windowed ** 2)  # P signal
    noise_power = np.mean(quantization_noise ** 2)
    SNR = 10 * np.log10(signal_power / noise_power)
```