25 Spring ECEN 610: Mixed-Signal Interfaces

Lab2: Signal to Noise Ratio, Quantization

Name: Yu-Hao Chen

UIN:435009528

Section:601

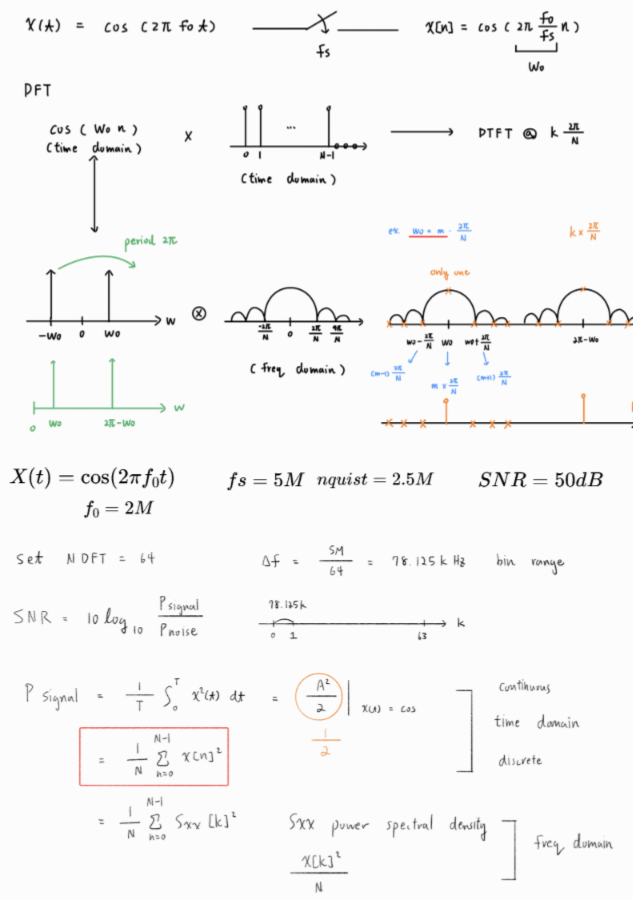
Professor: Sebastian Hoyos

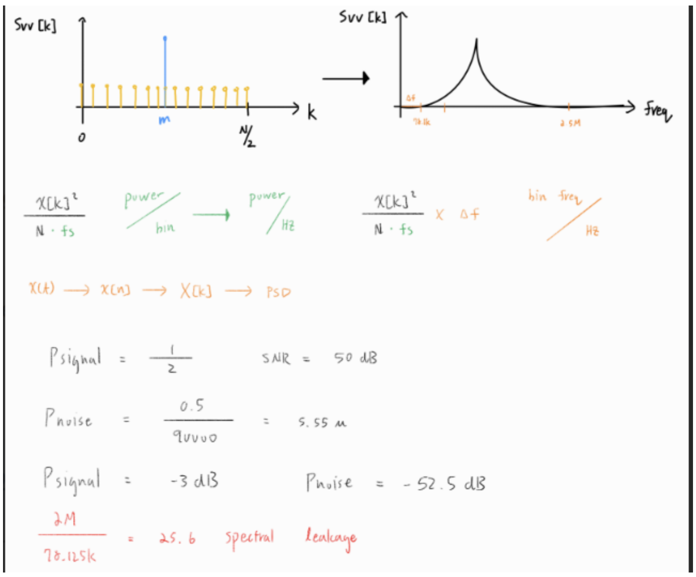
TA: Sky Zhao

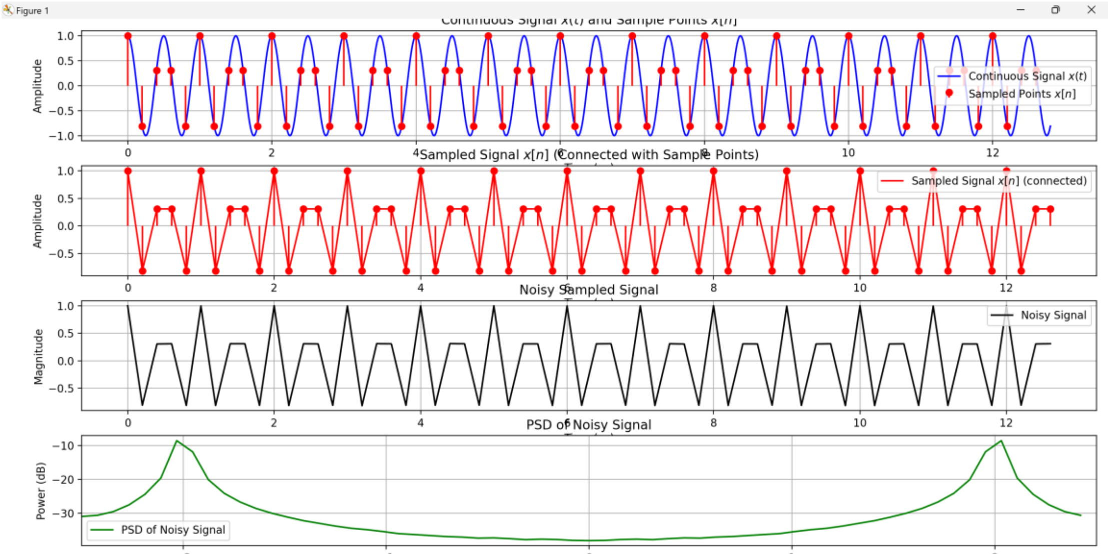
GitHub: https://github.com/Yu-HaoChen/TAMU_ECEN610_Mixed_signal/tree/main

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.

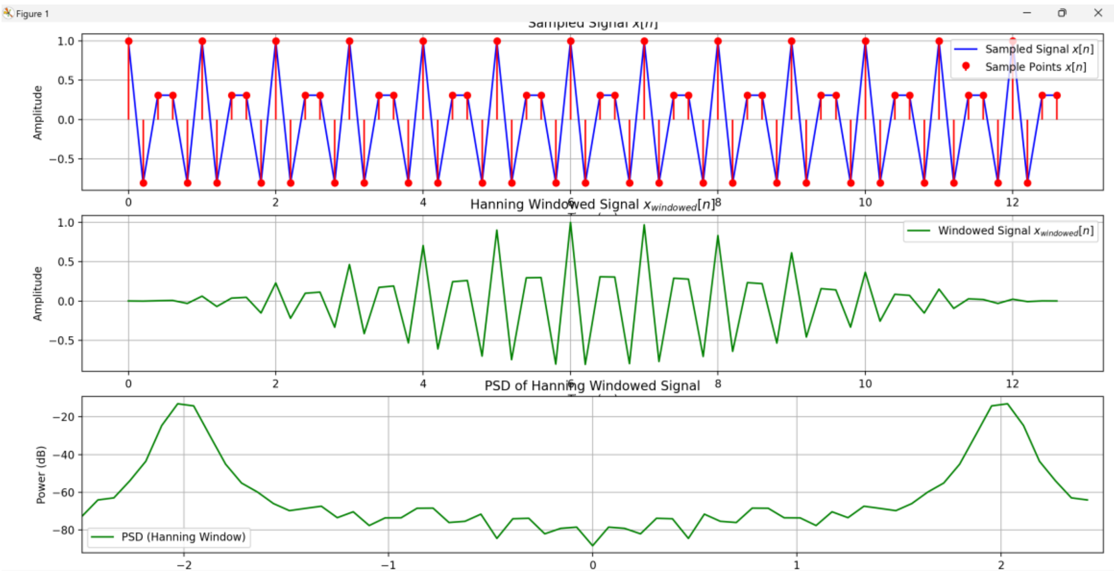
a) Add Gaussian noise to the sampled sinewave such that the signal SNR is 50 dB. Find first the variance of the Gaussian noise needed to produce the target SNR. Calculate and plot the Power Spectral Density (PSD) from the DFT of the noisy samples. Corroborate that the SNR calculation from the DFT plot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



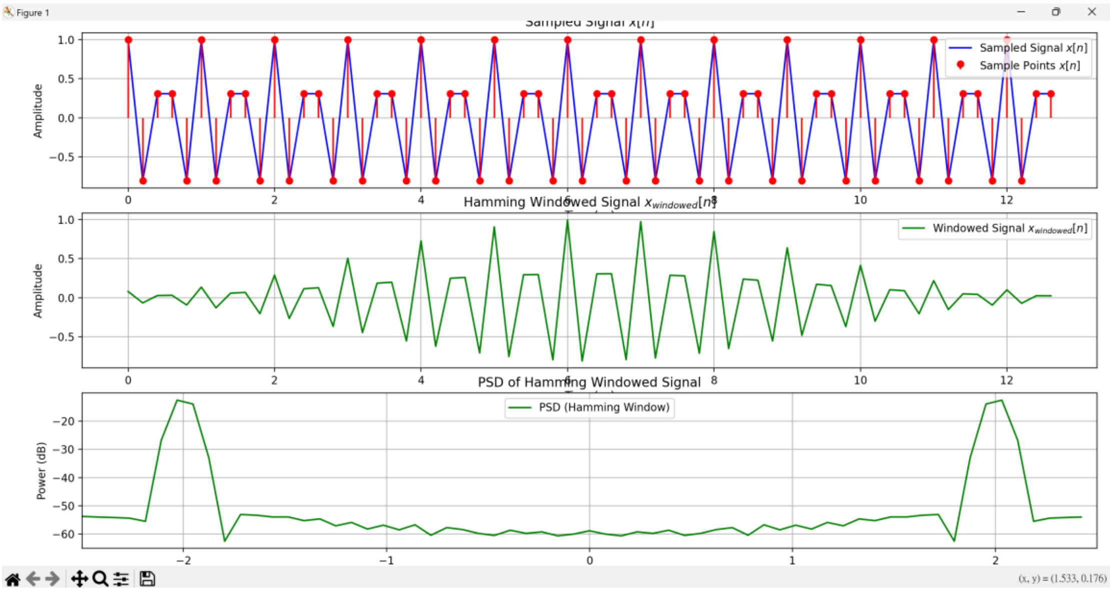




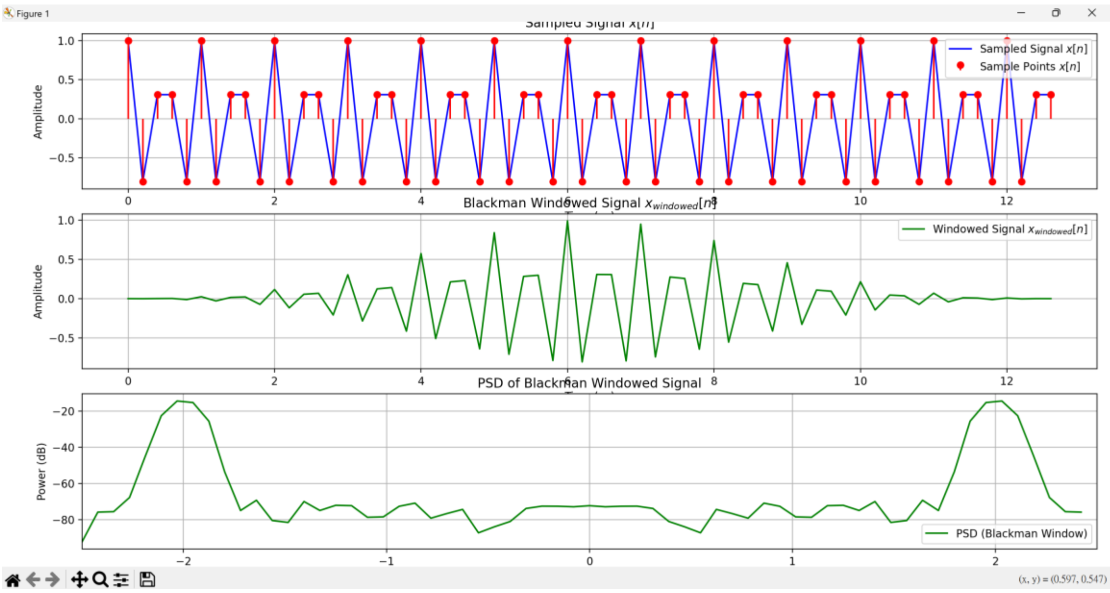
b) Now repeat a.) using a window before the DFT. Use the following windows: Hanning, Hamming, Blackman. What are your conclusions? NOTE: The use of windows mentioned above spreads the signal power. You must take this into account when computing SNR.



Hanning window



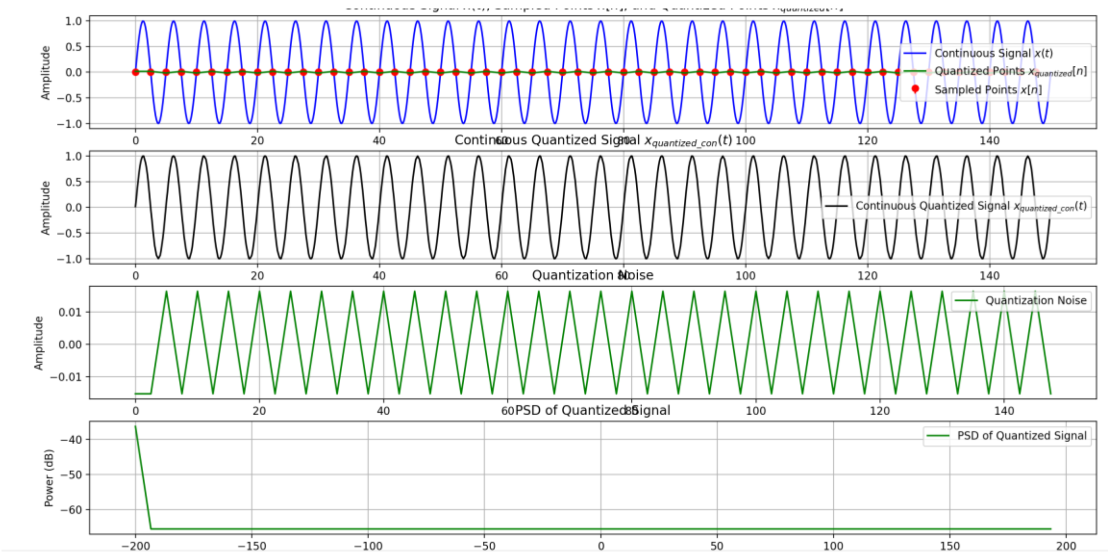
Hamming window



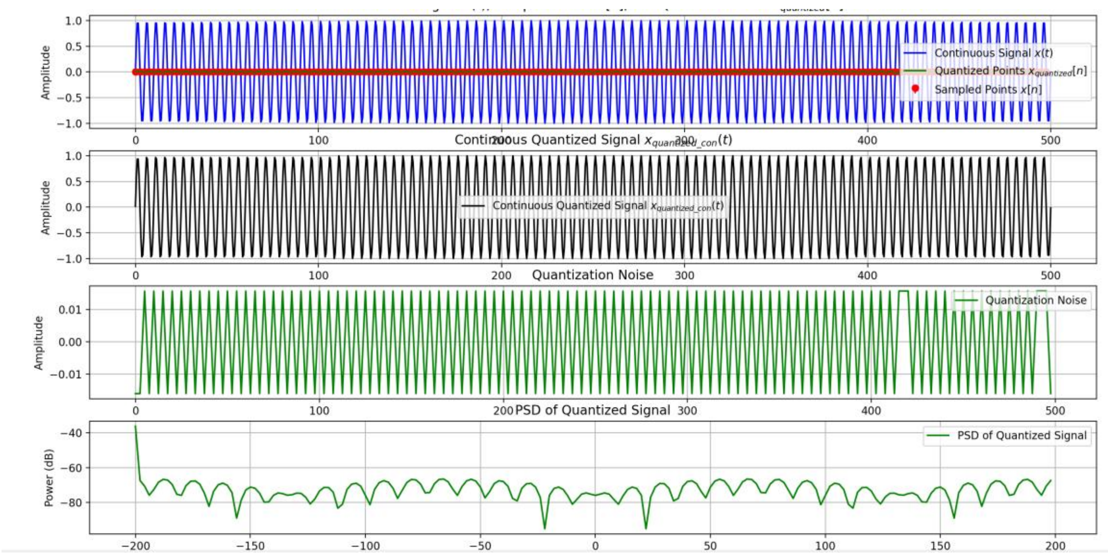
Blackman window

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?



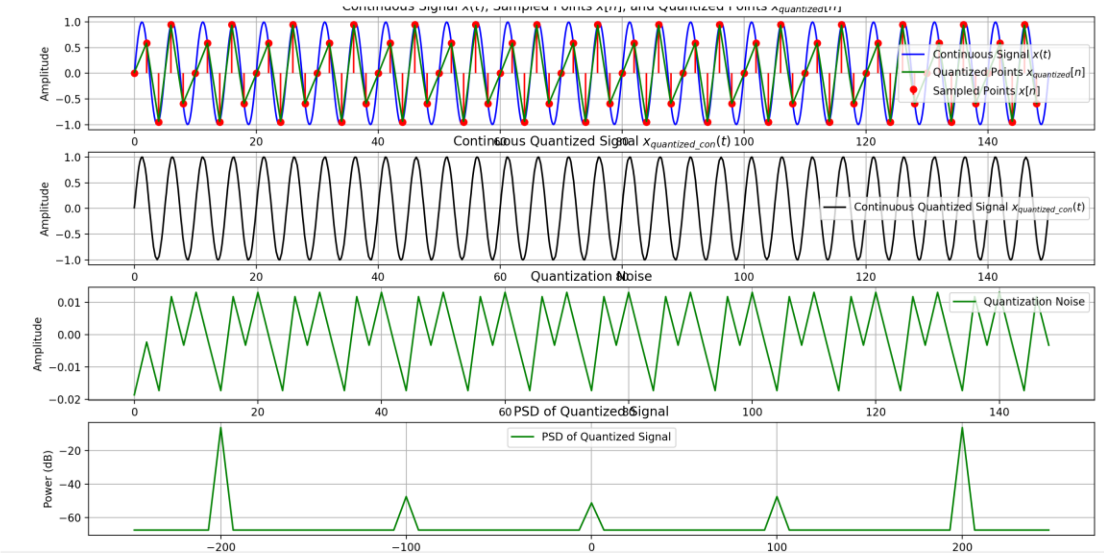

```
=== SNR for 30 Cycles ===
SNR = -240.12 dB
```




```
=== SNR for 100 Cycles ===
SNR = -229.76 dB
```

- The periodicity of quantization noise becomes most pronounced when the input frequency fin and the sampling rate fs form a simple integer ratio. This can cause significant variations in the measured SNR depending on the number of sampled cycles, the record length, and how the FFT bins are aligned.
- By breaking this periodicity—such as by slightly adjusting the input frequency, adding dither, or using an appropriate window and record length design—the quantization noise can be spread across the entire bandwidth. This makes the noise more closely resemble ideal "white noise," and thus yields an SNR measurement that is closer to the theoretical value.

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

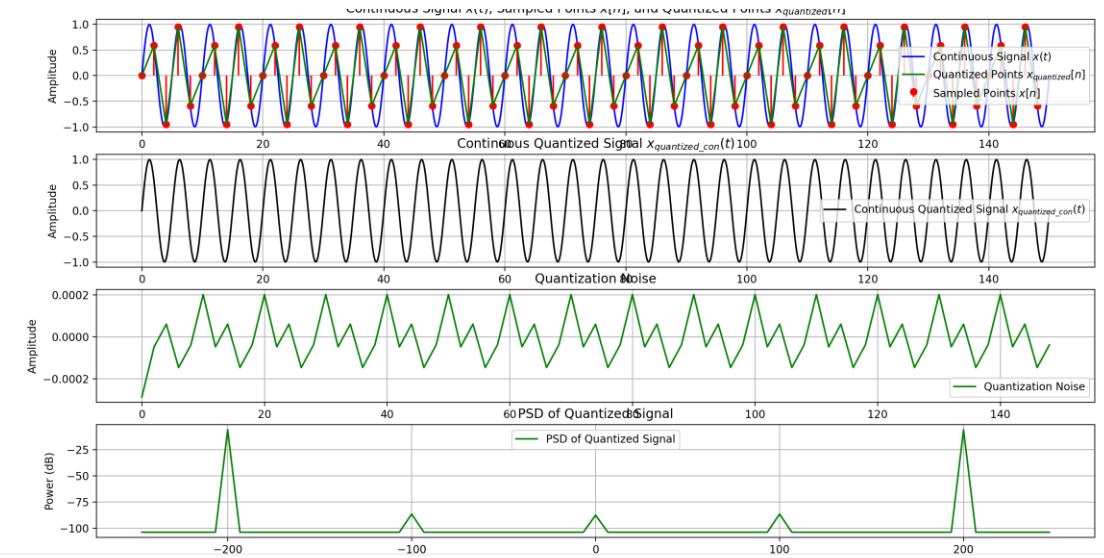



```
=== SNR for 30 Cycles ===
SNR = 35.68 dB
```

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?

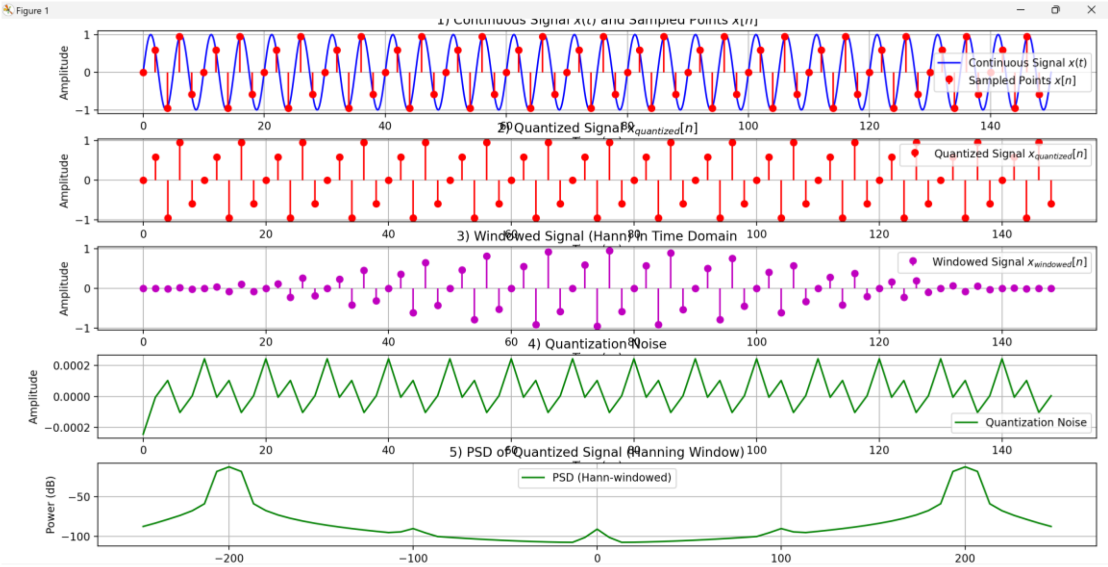
$$SNR = \frac{P_{signal}}{P_{noise}} = \frac{\frac{Vref^2}{2}}{\frac{\Delta V^2}{12}} = \frac{\frac{Vref^2}{2}}{\frac{\left(\frac{2Vref}{2^N}\right)^2}{12}} = \frac{\frac{Vref^2}{2}}{\frac{4Vref^2}{12 \cdot 2^{2N}}} = \frac{6 \cdot 2^{2N}}{4} = \frac{3}{2} \cdot 2^{2N}$$

$$10 \log_{10} \frac{3}{2} \cdot 2^{2N} = 10 \log_{10} \frac{3}{2} + 10 \cdot 2N \log_{10} 2 = 6N + 1.72$$




```
=== SNR for 30 Cycles ===
SNR = 74.89 dB
```

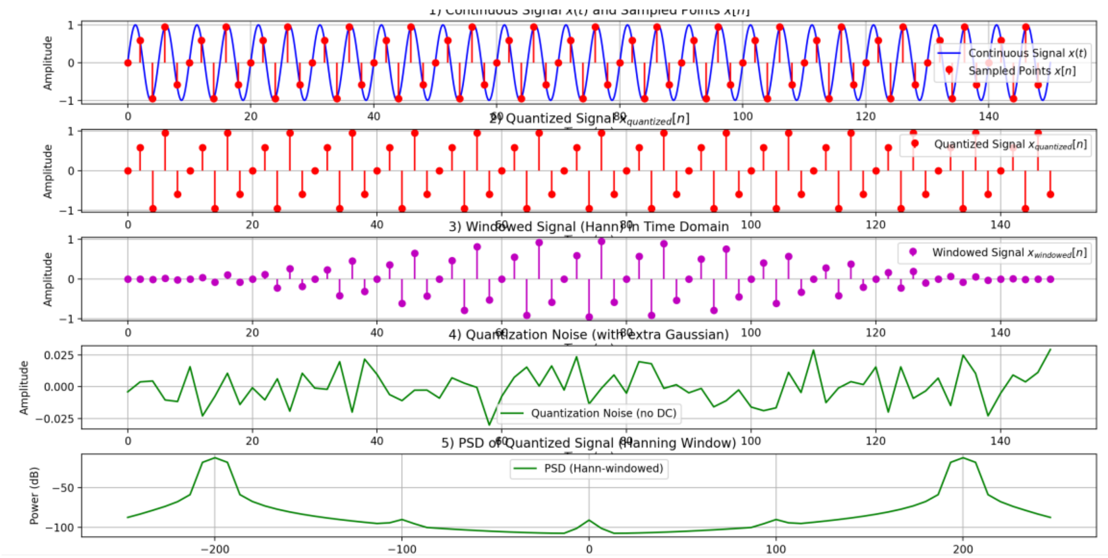
d) Use a Hanning window and repeat c). What is the SNR? Make your own conclusions.




```
=== SNR for 30 Cycles ===
SNR = 74.89 dB
```

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

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




```
=== Measured SNR ===
SNR = 34.33 dB
p_quant_error = 1.621523e-08
p_noise_total = 1.585000e-04
p_noise       = 1.584838e-04 ( = p_noise_total - p_quant_error )
```

■ 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

```python
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.sin(2 * np.pi * F * t_cont)  # x(t)
    x_input = Vref * np.sin(2 * np.pi * F * t_n)  # x[n]

    # === 2. Quantization ===
    x_quantized_con = np.round((x_input_cont + Vref) * (quantization_levels - 1) / (2 * Vref)) * (2 * Vref) / (quantization_levels - 1) - Vref
    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
    print(x_quantized)
    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)
```

Q2 window

```python
def simulate_quantization(N_cycles):
    N = int(N_cycles * Fs / F)
    t_n = np.arange(N) / Fs
    t_cont = np.linspace(0, N / Fs, 1000)

    x_input_cont = Vref * np.sin(2 * np.pi * F * t_cont)
    x_input = Vref * np.sin(2 * np.pi * F * t_n)

    x_quantized = np.round((x_input + Vref) * (quantization_levels - 1) / (2 * Vref)) \
                  * (2 * Vref) / (quantization_levels - 1) - Vref
    quantization_noise = x_input - x_quantized


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

    window = np.hanning(N)
    x_windowed = x_quantized * window

    X_k = np.fft.fft(x_windowed, 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 noise

```python
def simulate_quantization_5plots(N_cycles, p_noise_total=1.585e-4):
    # 1) Generate discrete-time signal x[n] for N integer cycles
    N = int(N_cycles * Fs / F)
    t_n = np.arange(N) / Fs
    t_cont = np.linspace(0, N / Fs, 1000)  # For continuous plotting

    # Create a clean sinusoidal input signal
    x_input_cont = Vref * np.sin(2 * np.pi * F * t_cont)  # continuous
    x_input = Vref * np.sin(2 * np.pi * F * t_n)          # discrete

    # 2) Quantize the clean signal
    x_quantized = np.round((x_input + Vref) * (quantization_levels - 1) / (2 * Vref)) \
                  * (2 * Vref) / (quantization_levels - 1) - Vref

    # 3) Compute original quantization error power
    p_quant_error = np.mean((x_input - x_quantized)**2)

    # 4) Compute p_noise = p_noise_total - p_quant_error
    p_noise = p_noise_total - p_quant_error
    if p_noise < 0:
        p_noise = 0.0
    sigma = np.sqrt(p_noise)

    # Generate Gaussian noise with variance = p_noise
    G_noise = np.random.normal(0, sigma, size=x_quantized.shape)

    # Final quantization noise = (x_input - x_quantized) + G_noise
    noise = (x_input - x_quantized) + G_noise
    noise_nodc = noise - np.mean(noise)

    # 5) Create a Hann window for the quantized signal
    window = np.hanning(N)
    x_windowed = x_quantized * window
```

```
# 5) Create a Hann window for the quantized signal
window = np.hanning(N)
x_windowed = x_quantized * window

# 6) Compute FFT & PSD (Hann-windowed)
X_k = np.fft.fft(x_windowed, N)
X_k_shifted = np.fft.fftshift(X_k)
freqs = np.fft.fftfreq(N, 1 / Fs)
freqs_shifted = np.fft.fftshift(freqs)

df = Fs / N
psd = (np.abs(X_k_shifted) ** 2 / (N * Fs)) * df
psd_dB = 10 * np.log10(psd + 1e-20)
```