Práctica 1: Sampling and Quantization

Renato Bedriñana Cárdenas Hugo Blanco Demelo

24 de octubre de 2025

1 Task 1

Question: Give your interpretation of the resulting graphs. Do the quantization levels correspond with the values you had expected?

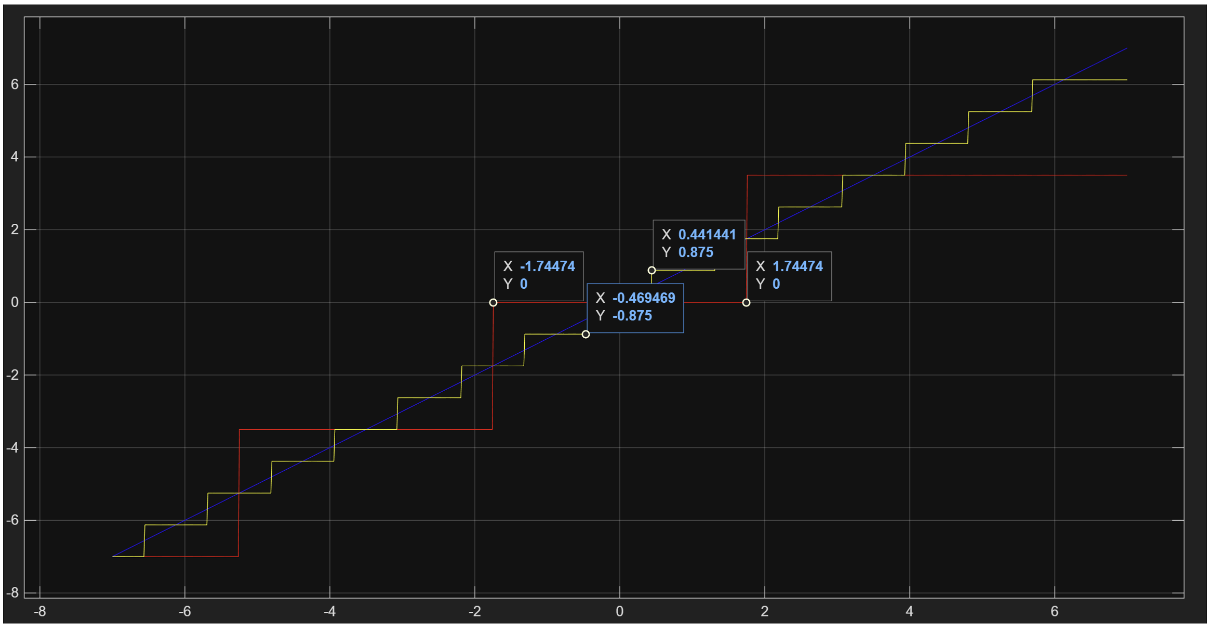
We represent the original continuous signal x (blue) and the 2 quantized version using 2 (red) and 4 (yellow) bits. As expected, the 2 bit quantization produces fewer discrete levels than the 4 bits quantization. Increase the number of bits decreases Δ, resulting in smaller steps and a quantized signal that follows the input more closely.

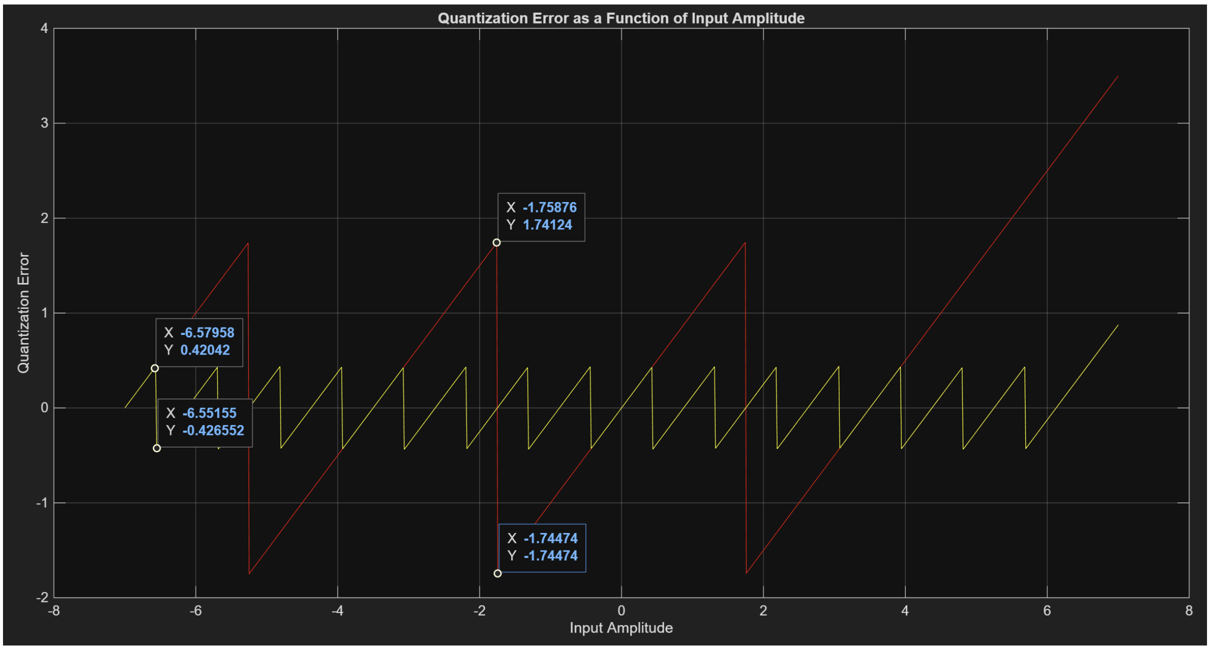
Question: For both cases, represent the quantization error as a function of input amplitude in the range $[-7, +7]$ and comment on your results. Is this error always within the $[-\Delta/2, +\Delta/2]$ interval?

The magnitude of the error decreases as the number of bits increases, since a smaller quantization step Δ reduces the maximum deviation between the input and its quantized version. The $[-\Delta/2, +\Delta/2]$ in each case is as follows:

- For N = 2 the Δ value we get is $\Delta = 3{,}5$, so the interval should be $[-1{,}75, 1{,}75]$.
- For N = 4 the Δ value we get is $\Delta = 0{,}875$, so the interval should be $[-0{,}4375, 0{,}4375]$.

In both cases, the error remains bounded within the theoretical interval $[-\Delta/2, +\Delta/2]$.

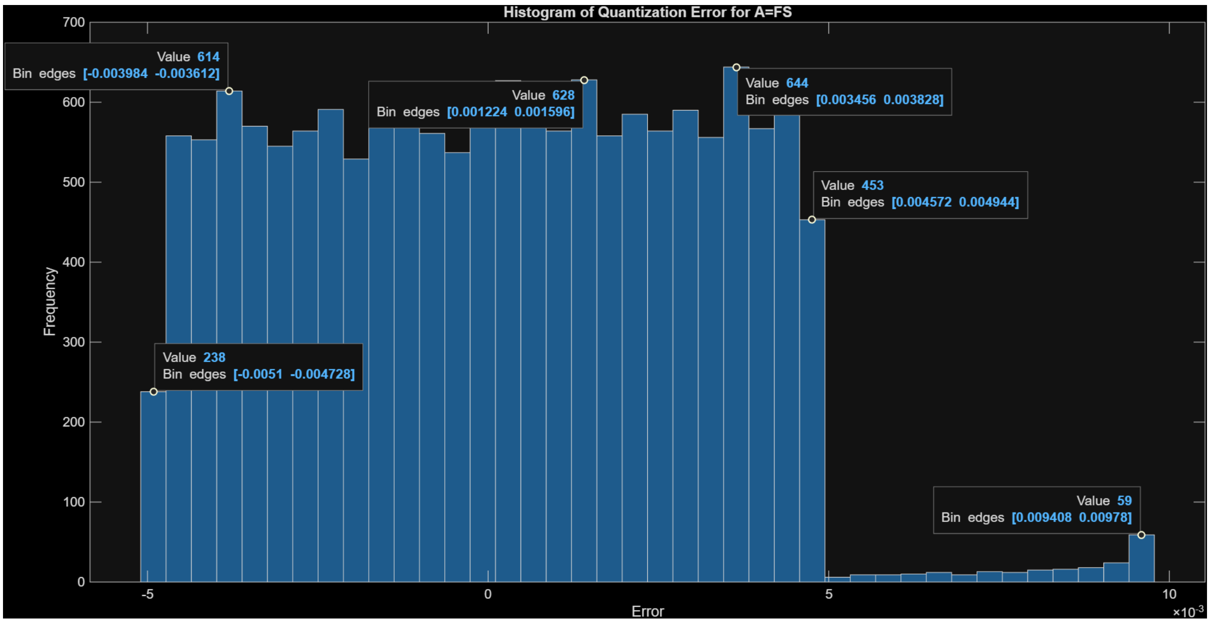




2 Task 2

Question: Assume a full-scale sinusoidal input and plot the histogram of the quantization error. Do you observe what you expected, or not?

Due we have an amplitude equal to FS we can expect clipping. We have $\Delta = \frac{2*FS}{2^N} = 0{,}0098$, the $[-\frac{\Delta}{2}, +\frac{\Delta}{2}]$ interval should be uniformly distributed (while the input does not get clipped) between $[-0{,}0049, +0{,}0049]$. In the histogram we can see that in that interval the error is uniformly distributed, but there is an error tail in the positive extreme. It means that there is **clipping** in the positive.



Question: Explain the operation of the Matlab command var. Estimate the variance of the quantization error using var, and compare it to its theoretical value. Estimate the value (in dB) of the Signal to-Quantization Noise Ratio (SQNR) and compare it to its theoretical value.

The MATLAB command `var` computes the variance of a set of values. For a vector x, it calculates: $\text{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$, where $\bar{x}$ is the mean of the values in x and n is the total number of samples. We used `var(x,1)` to compute the population variance (divide by n).

The empiral value of the variance of the quantization error we got is $8{,}72123e-06$, and the theoretical value is $\frac{\Delta^2}{12} = 7{,}94729e-06$.

The estimated value of the SQNR in dB we got is 61,5633 dB, and the theoretical value is 61,9597 dB.

Question: Repeat the previous steps for sinusoids with different amplitudes, and with decreasing resolutions of 12, 10, 8, 6 and 4 bits, in order to fill Table 1, rounding the SQNR values (in dB) to two decimal places. Comment on your results.

	$A = 0{,}5 \cdot \texttt{FS}$		$A = 0{,}75 \cdot \texttt{FS}$		$A = \texttt{FS}$		$A = 1{,}03 \cdot \texttt{FS}$	
	SQNR (dB)		SQNR (dB)		SQNR (dB)		SQNR (dB)	
N	theory	measured	theory	measured	theory	measured	theory	measured
12	67.98	68.03	71.5	71.54	74.00	73.7	74.26	38.47
10	55.94	56.01	59.46	59.53	61.96	61.56	62.22	38.19
8	43.90	44.04	47.42	47.53	49.92	49.15	50.18	36.97
6	31.86	32.13	35.38	35.6	37.88	36.52	38.14	32.27
4	19.82	20.37	23.34	23.78	25.8397	23.63	26.1	22.53

Cuadro 1: Pertaining to Task 2.

For amplitudes below FS (0.5*FS and 0.75*FS) the empirical SQNR values closely match the theoretical predictions. For an amplitude equal to FS, the empirical values still align well with theory, indicating minimal clipping effects. However, as the amplitude exceeds FS (1.03*FS), discrepancies arise due to clipping effects, SQNR collapses and even adding more bits does not solve the problem.

As the number of bits decreases the variance of the error grows roughly as expected and SQNR drops approximately 6 dB/bit. For moderate amplitudes the theory remains a good approximation down to mid-low N (but deviations increases as N gets smaller).

3 Task 3

Question: Suppose that you have an N-bit A/D converter with tunable FS, and you know that your input samples follow a symmetric triangular pdf in some interval $[-x_0, x_0]$. Intuitively, how would you set the FS value of your converter? What would the resulting rms value σ_x in dBFS be?

If you set $FS < x_0$ any imput $|x|$ greater than FS will be clipped. If $FS > x_0$,we would be wasting the converter's since the signal would never reach the limits. Therefore, the value of FS should be x_0.

To reach the variance of a symmetric triangular distribution we need to make some calculations:

$$Var(x) = E[x^2] - (E[x])^2 = E[x^2] + 0 = \int_{-x_0}^{x_0} x^2 f(x)dx = x_0^2/6$$

$\sigma_x = \sqrt{var(x)} = \frac{x_0}{\sqrt{6}}$ and in dBFS (with $x_0 = \text{FS}$) would be $20\log_{10}(1/\sqrt{6}) = -7,78$ dBFS.

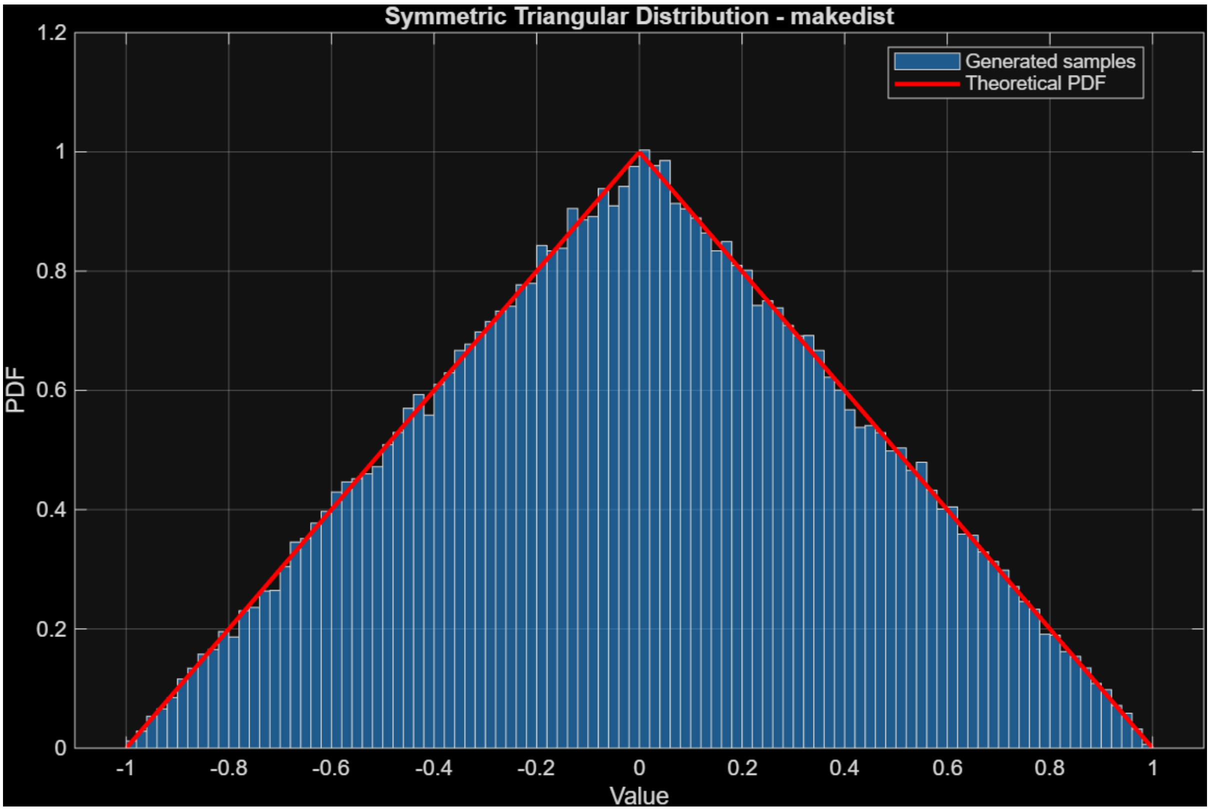
Question: Explain how to generate in Matlab samples of a random variable following a symmetric triangular pdf with zero mean and rms value σ_x. Check the histogram and use the commands mean and var to validate your approach

We have two options to do it:

- Option 1: The easiest way to generate a random variable with triangular pdf is using the function *makedist* from Matlab. The function spects the parameters A, B and C that define the triangular distribution. So we set the function parameters to get a symmetric triangular distribution centered at 0: `makedist('Triangular','A',-x0,'B',x0,'C',0)`

  The result of the mean and var commands are as follows:

  - Empirical mean: -4.58885e-05, wich is near to 0, very close to our target mean.
  - Empirical rms value [dBFS]: -7.78072, wich is very close to our target variance.



To do it, we can use the following code:

```
x0 = 2;
A = -x0; B = 0; C = +x0; % simetria = media 0
```

```
            pd = makedist('Triangular','A',A,'B',B,'C',C);
            N = 100000;
            samples = random(pd, N, 1);

            % comprobaciones rapidas
            emp_mean = mean(samples);
            emp_var = var(samples);
            emp_desv_std = std(samples);

            % valores teoricos
            % theo_mean = 0; % simetria centrado en 0
            theo_var = (A^2 + B^2 + C^2 - A*B - A*C - B*C)/18;
            rms = 20*log10(sqrt(theo_var)/x0);

            fprintf('Theorical mean: 0; emp mean: %.2f\n',emp_mean)
;
            fprintf('Theorical var: %.2f; emp var: %.2f\n',theo_var
,emp_var);
            fprintf('Sigma value: %.2f\n',sqrt(theo_var));
            fprintf('rms value in dBFS: %.2f\n',rms)

            % ver histograma y pdf teorica
            xgrid = linspace(A,C,500)';
            figure
            histogram(samples,100,'Normalization','pdf')
            hold on
            plot(xgrid, pdf(pd,xgrid), 'LineWidth',1.5)
            title('Triangular (media 0) -- muestras vs PDF')
            hold off
```

- Option 2: Another way we can generate samples of a random variable following a symmetric triangular pdf as the sum of two independent random variables X_1 and X_2 from a uniform distribution. When two independent random variables with uniform distributions are added, the resulting probability density function (PDF) becomes triangular. This can be understood both intuitively and mathematically.

  Intuitively, if each variable is uniform on $[-a, a]$, there are many pairs that sum near zero but only a few that produce sums near the extremes $\pm 2a$. Hence the PDF peaks at zero and decreases linearly towards the edges.

  Mathematically, let $Z = X_1 + X_2$ with X_1, X_2 independent and uniform on $[-a, a]$. The PDF of Z is the convolution of the two uniform PDFs:

$$f_Z(z) = (f_{X_1} * f_{X_2})(z) = \int_{-\infty}^{\infty} f_{X_1}(t)\, f_{X_2}(-t + z)\, dt.$$

  Carrying out the convolution yields the triangular PDF supported on $[-2a, 2a]$:

$$f_Z(z) = \frac{2a - |z|}{4a^2}, \qquad |z| \leq 2a.$$

  If we want the triangular distribution to have support $[-x_0, x_0]$, we must choose $a = x_0/2$. In that case the PDF simplifies to

$$f_Z(z) = \frac{x_0 - |z|}{x_0^2}, \qquad |z| \leq x_0,$$

Adding two uniform random variables with 0 mean, results in another random variable with 0 mean.

$$E[Z] = E[X_1 + X_2] = E[X_1] + E[X_2] = 0 + 0 = 0.$$

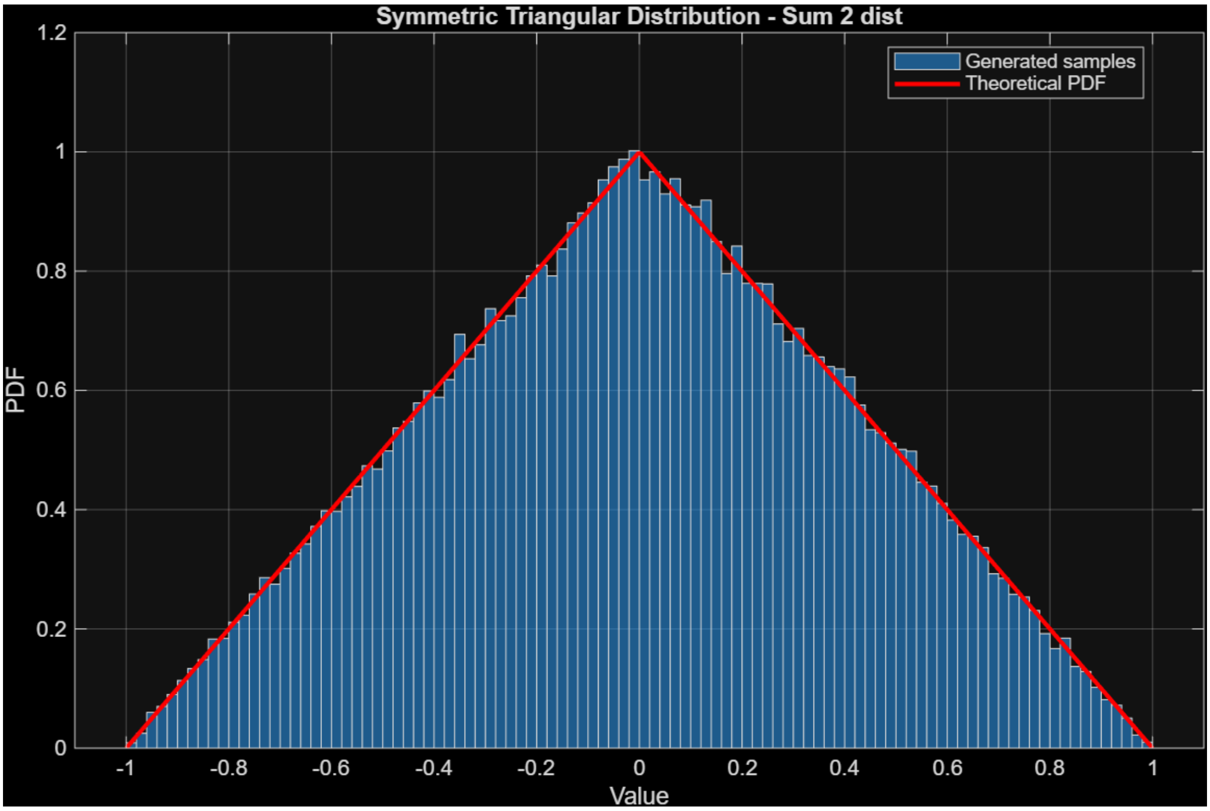
The variance of the sum of two independent random variables is the sum of their variances. So if we want a triangular distribution with variance σ_x (in dBFS), we need to set the variance of each uniform variable to $\sigma_x/2$.

$$Var(Z) = Var(X_1 + X_2) = Var(X_1) + Var(X_2) = \sigma_x/2 + \sigma_x/2 = \sigma_x.$$

For a uniform on $[-a, a]$ we have $\mathrm{Var}(X_i) = a^2/3$. Taking $a = x_0/2$ gives $\mathrm{Var}(Z) = 2 \cdot (x_0/2)^2/3 = x_0^2/6 = \sigma_x$, as required.

The result of the mean and var commands are as follows:

- Empirical mean: 0.00003, wich is close to 0, very close to our target mean.
- Empirical variance [dB]: -7.80701, wich is very close to our target variance.



we can doit as follows: REVISAR!!

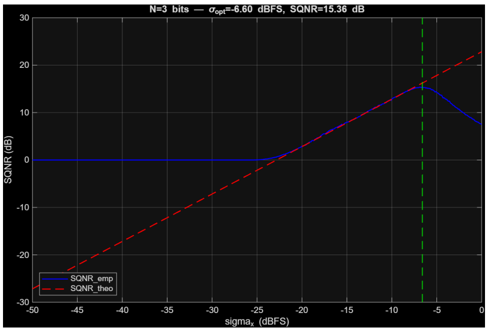
```
x0=2;
sigma0 = x0/sqrt(2);
N = 100000;

c = sigma0 * sqrt(3/2);
```

```
        x1 = (2 * rand(N, 1) - 1) * c;
        x2 = (2 * rand(N, 1) - 1) * c;

        y = x1 + x2;

        sample_mean = mean(y);
        sample_var = var(y);
        sample_rms = std(y);

        fprintf('--- Validation ---\n');
        fprintf('Target Mean: 0.0\n');
        fprintf('Sample Mean: %f\n\n', sample_mean);

        fprintf('Target Variance (sigma0^2): %f\n', sigma0^2);
        fprintf('Sample Variance: %f\n\n', sample_var);

        fprintf('Target RMS (sigma0): %f\n', sigma0);
        fprintf('Sample RMS: %f\n\n', sample_rms);

        figure;
        histogram(y, 100, 'Normalization', 'pdf', 'DisplayName'
, 'Generated Samples');
        grid on;
        hold on;

        a = 2*c;
        x_pdf = linspace(-a, a, 400);
        y_pdf = (1/a) * (1 - abs(x_pdf)/a);
        plot(x_pdf, y_pdf, 'r-', 'LineWidth', 2.5, 'DisplayName
', 'Theoretical PDF');

        title('Symmetric Triangular Distribution');
        xlabel('Random Variable Value');
        ylabel('Probability Density Function (PDF)');
        legend;
        hold off;
```

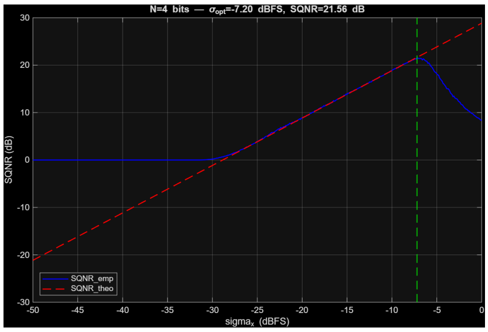
Question: Take $10 \cdot 2^{10}$ of these triangularly distributed samples, quantize them, and estimate the SQNR empirically for $N =$ 3, 4, 5 and 6 bits. Do this for σ_x varying in the range $[-50, 0]$ dBFS and in steps of $0{,}1$ dBFS. Plot the resulting curves (SQNR in dB vs. σ_x in dBFS) along with the theoretical expression

$$\mathrm{SQNR} = 6{,}02N + 4{,}77 - 20\log_{10}\frac{\mathrm{FS}}{\sigma_x} \qquad \textbf{(dB).} \tag{1}$$

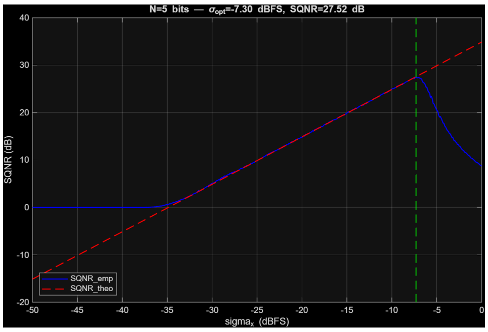
Are there any differences between the theoretical and empirical curves? If so, how do you explain them?



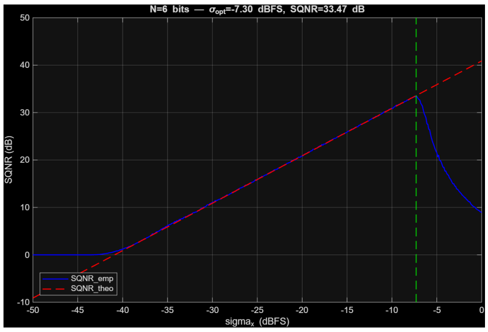
(a) N=3 bits - σ_{opt}=-6.60 dBFS, SQNR=15.36 dB



(b) N=4 bits - σ_{opt}=-7.20 dBFS, SQNR=21.56 dB



(c) N=5 bits - σ_{opt}=-7.30 dBFS, SQNR=27.52 dB



(d) N=6 bits - σ_{opt}=-7.30 dBFS, SQNR=33.47 dB

Figura 1: SQNR vs σ_x (dBFS) for triangularly distributed input at different quantization resolutions.

The comparison between theoretical and empirical SQNR curves for triangularly distributed inputs is shown in Figure 1. The red line represents the theoretical SQNR curve, while the blue line represents the empirical SQNR values obtained from quantizing the triangularly distributed samples.

Empirical curves deviate from the straight theoretical line for two practical reasons. At very small σ_x values, quantization noise is no longer uniformly distributed and we have no enough bits for quantization. At large σ_x values, clipping occurs, distorting the signal and reducing SQNR below theoretical predictions. Otherweise, in the mid-range of σ_x values, empirical results closely follow theoretical expectations and reaches the maximum SQNR (marked in the description of each image).

When we increase the number of bits N, the curve starts to follow the theorical curve with smaller values of σ_x. We reach a point where optimum σ_x value (where SQNR is maximized) approaches the theoretical value of -7.78 dBFS calculated earlier.

Question: In view of your results, what are the optimum values (regarding SQNR) of σ_x (in dBFS), and for the different resolutions analyzed (3 to 6 bits)? Does this agree with your intuition (see first point above)?

We can see in the previous plots 1 (see the green lines) that the optimum values of σ_x (where SQNR is maximized) for different resolutions are:

- N=3 bits: σ_{opt} = -6.60 dBFS
- N=4 bits: σ_{opt} = -7.20 dBFS

- N=5 bits: σ_{opt} = -7.30 dBFS
- N=6 bits: σ_{opt} = -7.30 dBFS

Those are the points where the empirical SQNR reaches its maximum value and then starts to decrease.

The value we calculated in the first point was -7.78 dBFS, which is close to the optimum values we obtained empirically. If we increase the number of bits, the optimum value gets closer.

Question: Repeat the previous points, but now using normally distributed input samples with zero mean and standard deviation σ_x.

In a Gaussian distribution, we can not set the $[-x_0, x_0]$ limit as in the triangular distribution. So it is a parameter that we can not control and clipping will always occur for some samples no matter how we set FS. But, we can still set FS to optimize SQNR. Following the definition of dBFS, we can set FS to be some multiple of σ_x.

$$\sigma_x = 20\log_{10}\frac{\sigma_x}{\text{FS}} = -20\log_{10} k \implies FS = k \cdot \sigma_x$$

To decide which k value is the best, we have to see the clip probability for each k. To express the clipping probability in terms of the standard normal CDF, we normalize the Gaussian variable as $Z = X/\sigma_x$, so that $Z \sim \mathcal{N}(0, 1)$. Then:

$$P(X > FS) = P\left(Z > \frac{FS}{\sigma_x}\right) = 1 - \Phi(k)\,,$$

where $\Phi(k)$ is the cumulative distribution function of the standard normal distribution. Therefore, considering both tails, the total clipping probability is:

$$p_{\text{clip}} = 2(1 - \Phi(k)), \quad \text{with } k = \frac{FS}{\sigma_x}.$$

So we have the next posible values for k:

- k = 1: σ_x = 0 dBFS (clipping prob = 31.73 %)
- k = 2: σ_x = -6.02 dBFS (clipping prob = 4.55 %)
- k = 3: σ_x = -9.54 dBFS (clipping prob = 0.27 %)
- k = 4: σ_x = -12.04 dBFS (clipping prob = 0.000063 %)

A good trade-off between clipping probability and SQNR can be achieved with k = 3, which gives a good compromise between dynamic range usage and distortion.

To generate normally distributed samples we can use the Matlab command `randn`, which generates samples from a standard normal distribution (mean 0, variance 1). And then we scale the samples to get the desired standard deviation σ_x. The output plot is visible in the Figure 2.

The result of the mean and var commands are as follows:

- Empirical mean: -0.000265784, close to our target mean.
- Empirical variance [dB]: -9.56729, again, close to the value we expected.

Analyzing the SQNR vs σ_x curves in different N levels, we obtain similar outputs as the triangular distribution. We can check them in the Figure 3, we observe that the empirical SQNR values deviate from the theoretical predictions, especially at higher σ_x values.

But we need more level of σ_x to reach the optimum point, because of the clipping that occurs in the Gaussian distribution, and we get a fewer maximum SQNR value than in the triangular distribution. This is produced because the Gaussian distribution has heavier tails, leading to more frequent clipping events at higher σ_x levels.

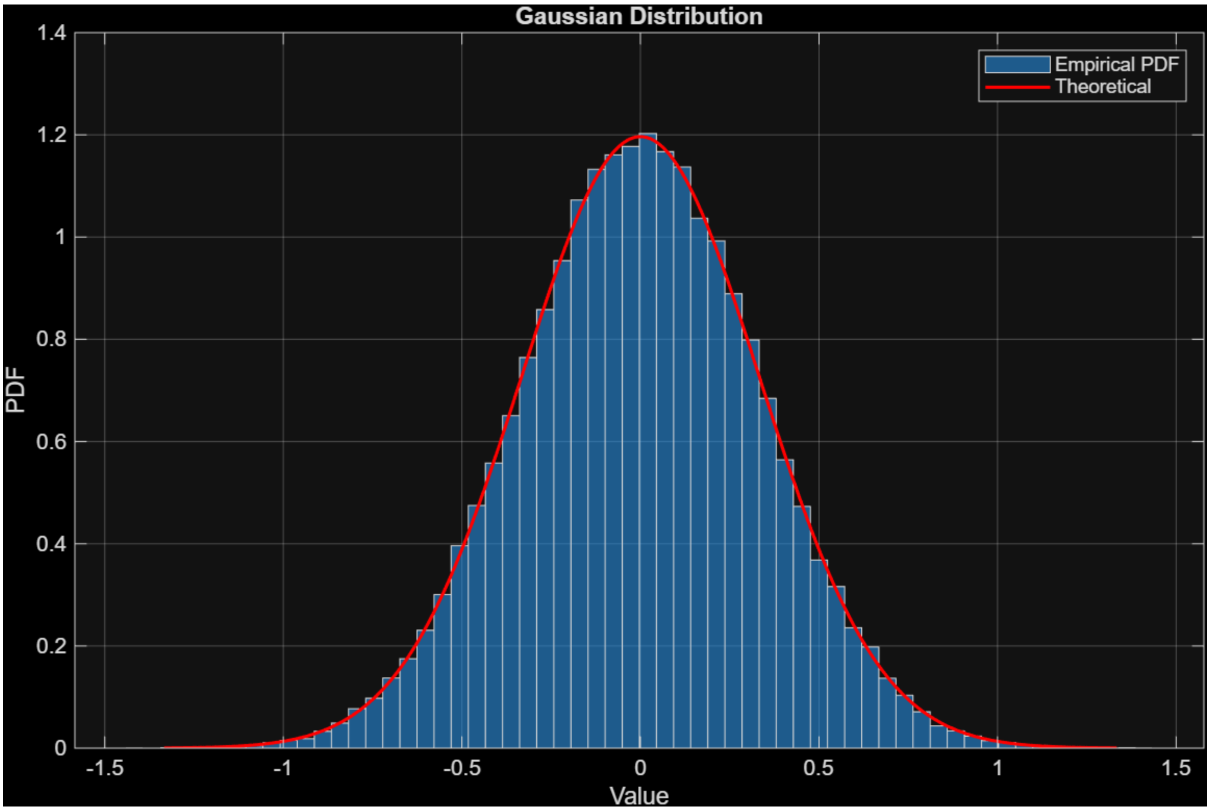
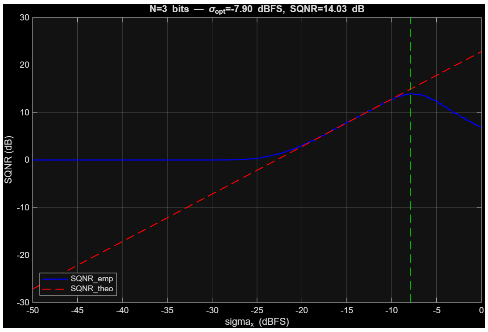
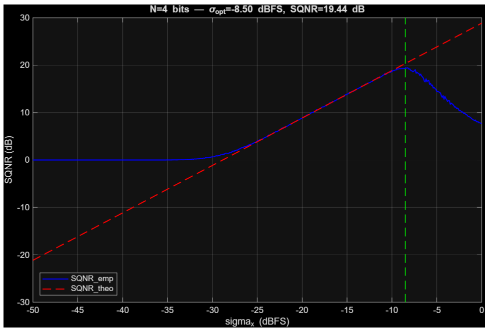


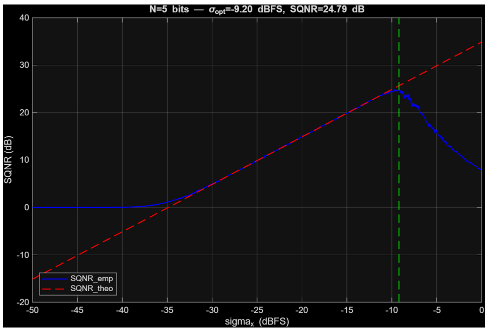
Figura 2: Gaussian distribution (normalized)



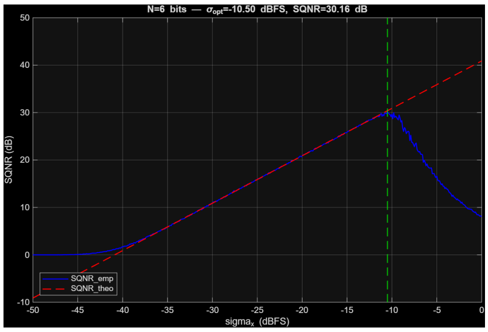
(a) N=3 bits - σ_{opt}=-7.90 dBFS, SQNR=14.03 dB



(b) N=4 bits - σ_{opt}=-8.50 dBFS, SQNR=19.44 dB



(c) N=5 bits - σ_{opt}=-9.20 dBFS, SQNR=24.79 dB



(d) N=6 bits - σ_{opt}=-10.50 dBFS, SQNR=30.16 dB

Figura 3: SQNR vs σ_x (dBFS) for normal distribution input at different quantization resolutions.

4 Task 4

Question: Assume a full-scale sinusoidal input with $f_0 = 37{,}1094 MHz$, and let the FFT size be M = 1024. Generate $15 \cdot M$ samples of $x(t)$ (at fs = 100 MHz) and quantize them to N = 12 bits. Break the vector xq of quantized samples into 15 size-M blocks using, e.g., the command reshape:

```
xqblocks = reshape(xq, M, 15);
```

so that each column of the $M \times 15$ matrix xqblocks will contain the corresponding block of size M . Now, since the fft command computes the FFT columnwise, in order to apply an M -point FFT to each block, we simply make

```
X = fft(xqblocks, M);
```

Average the squared magnitude of the DFT coefficients over the 15 blocks and plot the results between 0 and fs/2, in dBFS. Observe the location and peak value of the principal frequency component, as well as the value of the noise floor. Do your observations agree (quantitatively) with what you would expect?

4.1 Theoretical values

First, we need to calculate the expected theoretical values for the signal peak and for the noise floor value.

4.1.1. Signal Peak

We have $f_0 = 37{,}1094MHz$ and $f_s = 100MHz$. As $f_0 < f_s/2$ we don't have aliasing. Terefore, we expect a signal peak at f_0, with a value of 0 DBFS, as it is a full-scale signal.

4.1.2. Noise Floor

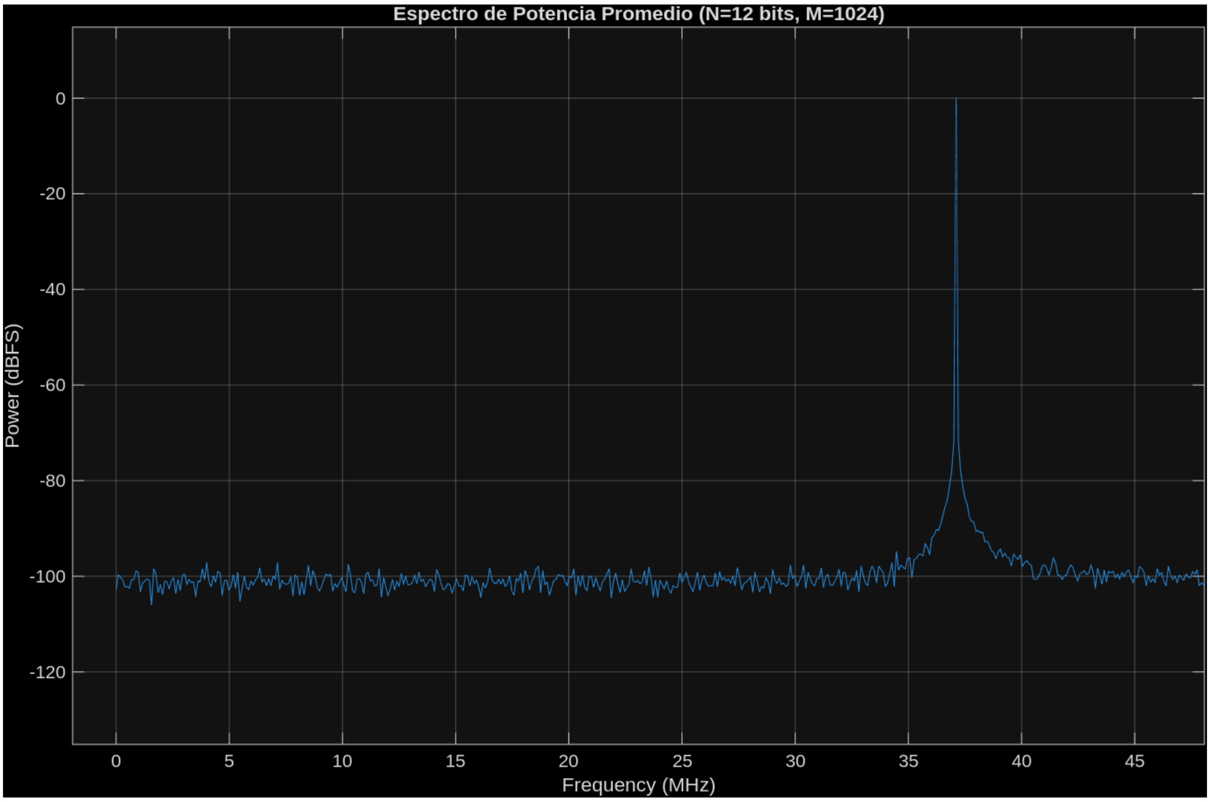
To calculate the theoretical SQNR we have the formula $SQNR = 6{,}02N + 4{,}77 - 20log_{10}(FS/\sigma_x)$ As we have a full scale sinusoid we have $\sigma_x = A/\sqrt{2} = FS/\sqrt{2}$ So, for $N = 12$ and $\sigma_x = FS/\sqrt{2}$ we have $SQNR = 73{,}99DBFS$

We have to calculate the processing gain, with the formula $10log_{10}(M/2)$. For $M = 1024$, we have a gain of 27.09 DBFS.

The noise floor will be $-(73{,}99 + 27{,}09) = 101{,}08DBFS$

4.2 Matlab execution

Executing the task_4_1.m matlab script we can see the next figure.



On the figure we can see a signal peak at 37.1094 MHz, whith a value of 0 DBFS.

This agrees quantitatively with the theory, which predicts a peak at the input frequency.

$f_0 = 37{,}1094MHz$ and a level of 0 dBFS due to the normalization used for a full-scale signal.

We can also see that the noise floor is around the 100 DBFS, which agrees with the theoretical value.

Question: Repeat the previous steps for an FFT size M = 256.

4.3 Theoretical values

As the frequency f_0 is the same, we would also have a signal peak on that point. Since is full-scale too, thew value of the peak would also be 0 DBFS.

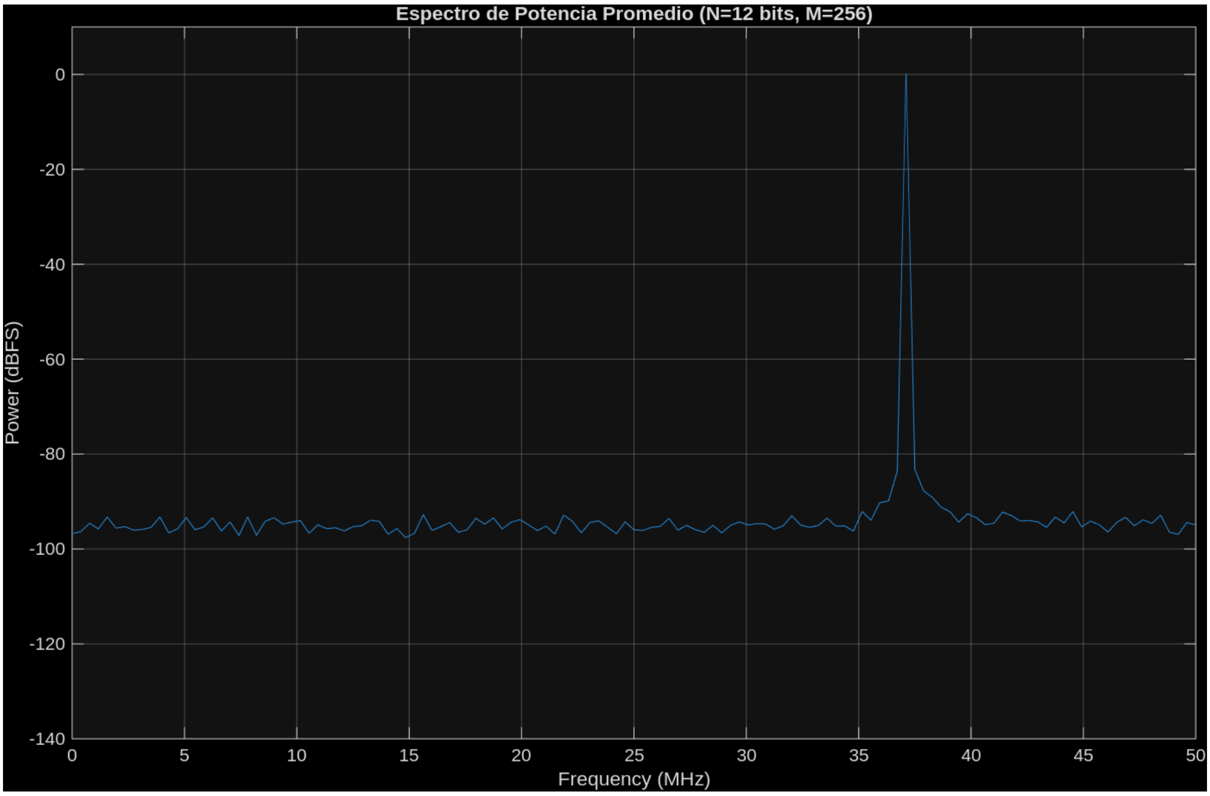
The SQNR will be the same, because we have the same number of bits and the same σ_x.

However, the gain will change, as we have a different value for M. $Gain = 10log_{10}(M/2) = 10log_{10}(256/2) = 21{,}07$

For M = 256 we will have a nois floor of 95.06 DBFS

4.4 Matlab execution

Executing the task_4_2.m script, we can see a noise peak of 0 DBFS at f_0 and a level of noise floor of approximately 95 DBFS



Question: Set again M = 1024, and repeat the analysis for decreasing resolutions of 10, 8 and 6 bits.

4.5 Theoretical values

As the frequency f_0 is the same, we would also have a signal peak on that point. Since is full-scale too, thew value of the peak would also be 0 DBFS.

The Gain will be the same as on task4_1 because we have thje same M.

The SQNR will change, because we have different values for N:

- N = 10: $SQNR = 6{,}02N + 4{,}77 - 20log_{10}(FS/\sigma_x) = 6{,}02 * 10 + 4{,}77 - 20log_{10}(\sqrt{2}) = 61{,}95DBFS$

- N = 8: $SQNR = 6{,}02N + 4{,}77 - 20log_{10}(FS/\sigma_x) = 6{,}02 * 8 + 4{,}77 - 20log_{10}(\sqrt{2}) = 49{,}91DBFS$
- N = 6: $SQNR = 6{,}02N + 4{,}77 - 20log_{10}(FS/\sigma_x) = 6{,}02 * 6 + 4{,}77 - 20log_{10}(\sqrt{2}) = 37{,}87DBFS$

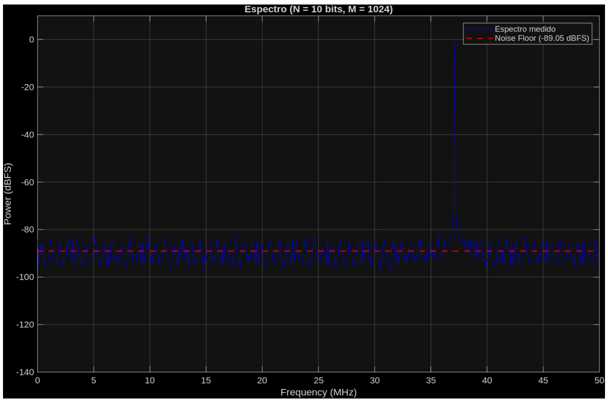
So the noise floor for each value of N will be:

- N = 10: $-(61{,}95 + 27{,}09) = -89{,}04DBFS$
- N = 8: $-(49{,}91 + 27{,}09) = -77DBFS$
- N = 6: $-(37{,}87 + 27{,}09) = -64{,}96DBFS$

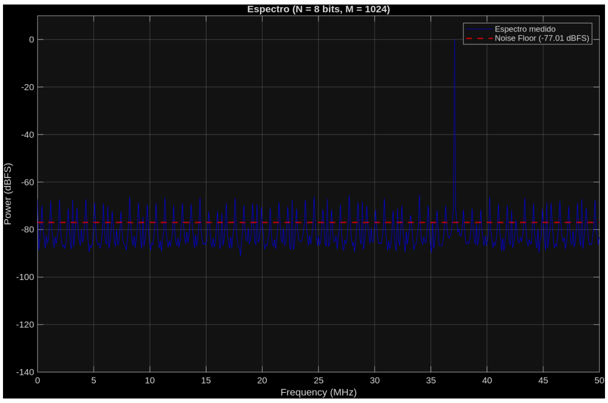
4.6 Matlab execution

Executing the the task_4_3.m script, we can see three figures with a peak of 0 DBFS on f_0. We also see a noise floor value of :

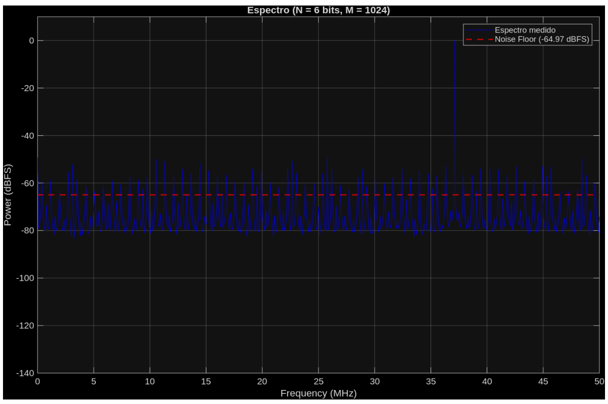
- N = 10: $-89{,}05DBFS$
- N = 8: $-77{,}01DBFS$
- N = 6: $-64{,}97DBFS$



(a) N=10 bits



(b) N=8 bits



(c) N=6 bits

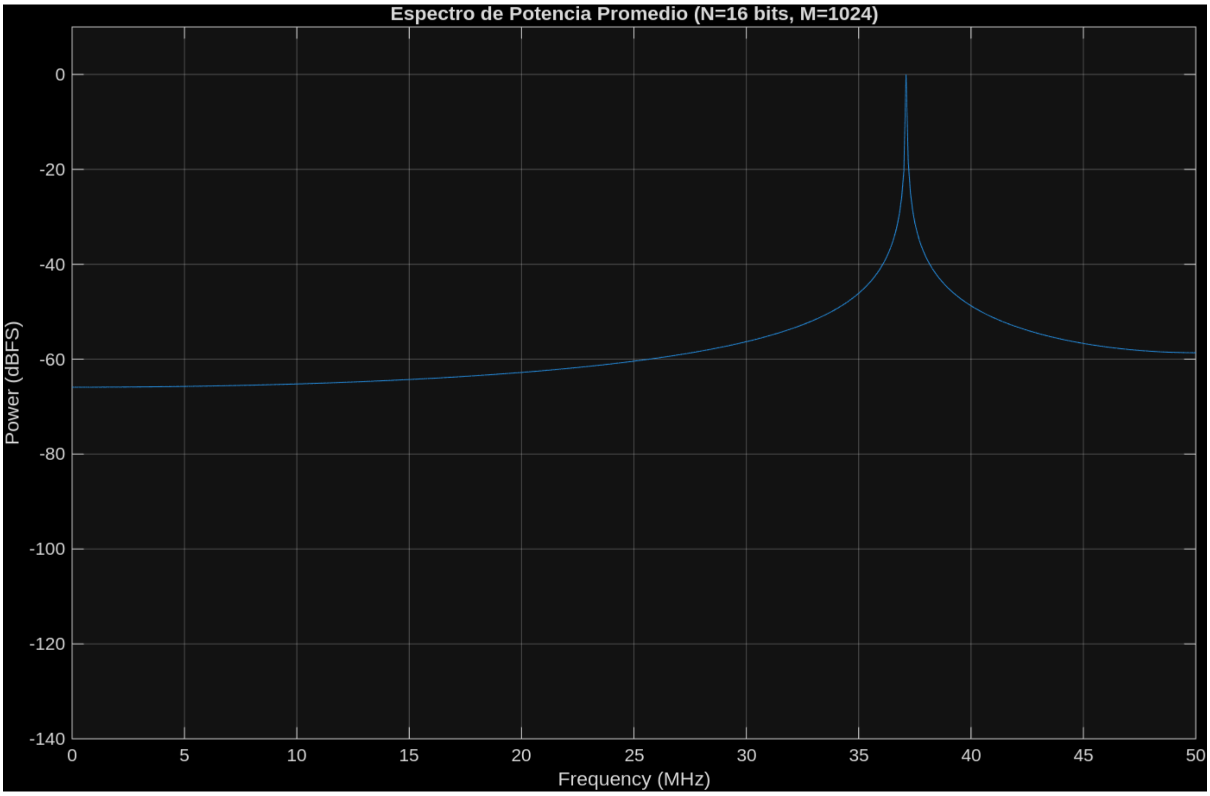
Question: Consider again M = 1024 and N = 12 bits. Repeat the analysis reducing the amplitude of the sinusoid to 1/3 of the full scale value, and compare your observations with the theoretical prediction.

4.7 Theoretical values

For a siusoid of amplitude $A = \alpha * FS, \alpha < 1$, we have $SQNR = 6{,}02N + 1, 76 + 20log10(A)$. For $A = 1/3$ we have a SQNR of 64.81 DBFS. The gain will be the same, so we will have a noise floor of 91.9 DBFS

Question: Let M = 1024, N = 12 bits and a full-scale sinusoid. Slightly change the frequency of the sinusoid to 37.12 MHz and repeat the analysis. How do your observations change? Does it make any difference if you use a larger number of samples, say 100 * M ? What happens if you increase the resolution to 16 bits? How do you explain all these?

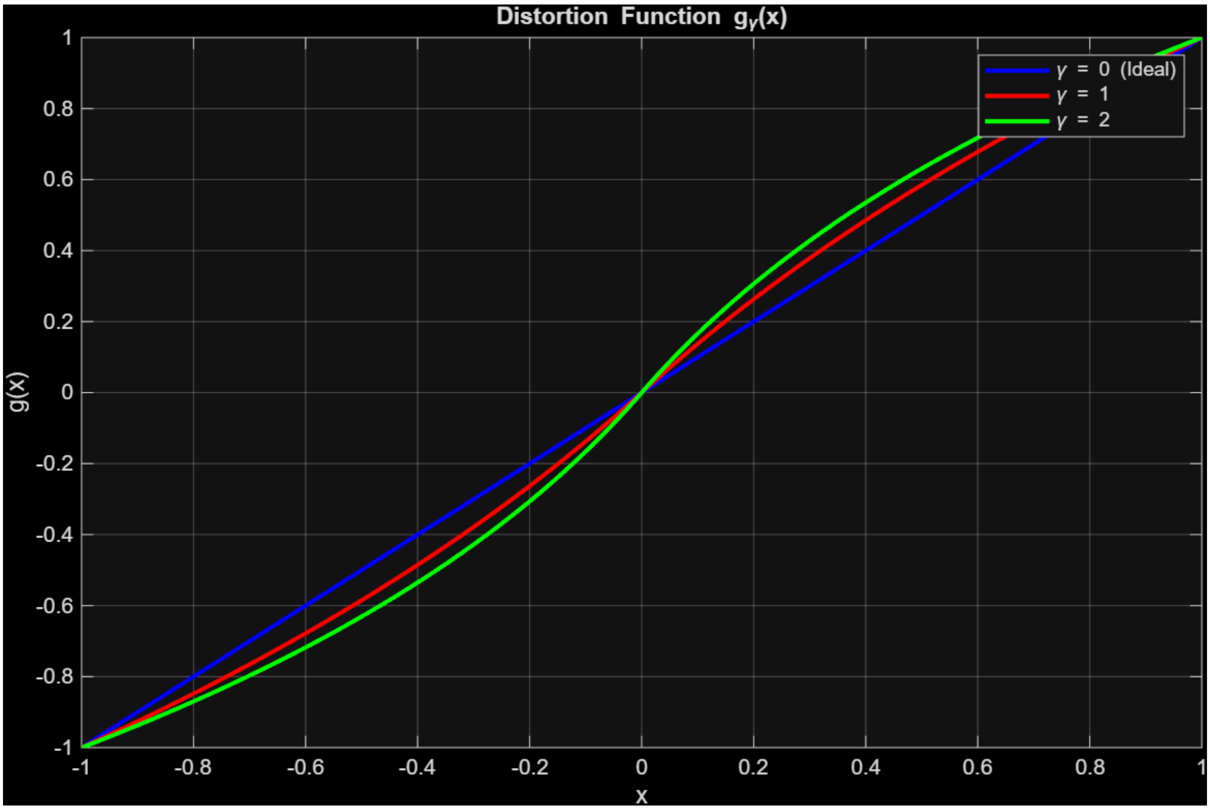
The f_0 we had, was a k of the FFT, so all the energy was on a single k: $k = \frac{f_0 * M}{fs} = \frac{37{,}1094 * 10^6 * 1024}{100 * 10^6} = 380$ When using the new frequency, the energy is between $k = 380$ and $k = 381$, so we see this power leak.



Increasing the number of samples or the resolution will not have effect, because we need to have the signal on a k. To do this, we can change either the f_0, the fs or M.

5 Task 5

Question: Plot $g_\gamma(x)$ vs. x in the range $x \in [-\mathrm{FS}, \mathrm{FS}]$ for $\gamma = 0$, 1 and 2. For input signals whose values are always much smaller than FS (in absolute value), what will be the effect of the nonlinearity?



Question: Modify the code in `quanti.m` and write a Matlab function `dquanti.m` implementing this nonuniform quantizer. The format should be similar to that of `quanti.m`, but including an additional input parameter `gama`:

```
xq = dquanti( x, FS, Nbits, gama );
```

We change the code as follows:

```
function y = dquanti(x, FS, Nbits, gama)
if gama == 0
    g_x = x; % Si gama=0, no hay distorsion (g(x) = x)
else
    g_x = sign(x) .* (FS / log(1 + gama)) .* log(1 + gama .* abs(x)
/ FS);
    g_x(x == 0) = 0;
end

FS    = abs(FS);
FSbin = 2^(Nbits-1);
LSB   = FS/FSbin;

    y = round(g_x/LSB);
    y = min(y, FSbin-1);
    y = max(y, -FSbin);
    y = y * LSB;
```

Question: Generate samples (at 100 MHz) of a full-scale sinusoid with $f_0 = 6{,}8359$ MHz. Quantize them to $N = 11$ bits using $\gamma = 0{,}003$ in `dquanti`. Determine the SFDR in dBFS using an FFT size $M = 2048$, and then with $M = 512$. Does the SFDR depend on the FFT size? Does the noise floor depend on the FFT size? How do you explain this?

xd

Question: Using $M = 2048$, repeat the previous step for $\gamma = 0{,}01$ and $0{,}1$. Are the spectral spurs located where you would expect?

xd

Question: Set now the amplitude to $\frac{\mathrm{FS}}{3}$. Using $M = 2048$, measure the SFDR and express it in both dBFS and dBc for $\gamma = 0{,}005$, $0{,}05$ and $0{,}1$. Will these values change if you repeat the analysis with $M = 512$?

xd

Question: Consider now samples (at 100 MHz and with 11-bit resolution) of a sinusoid with frequency $3{,}3202$ MHz and amplitude $\frac{\mathrm{FS}}{2}$. Obtain the THD for this nonuniform ADC with $\gamma = 0{,}3$ under the IEEE 1241-2000 specification, expressed in both dB and percentage.

xd

6 Task 6

Question: If the rms value of the aperture jitter is 20 ps, and the input signal is a full-scale sinusoid with frequency f_c, for which values of f_c will the aperture error power dominate the quantization noise power?

$SQNR = 20 * log_{10}(1/(2 * \pi * fc * \sigma))$
$fc = 1/(10^(SQNR/20) * 2 * \pi * \sigma$
fc should be greater than $1/(10^(SQNR/20) * 2 * \pi * \sigma$

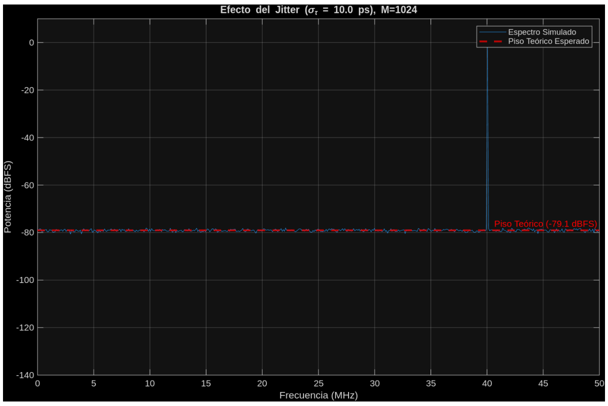
Question: If the rms value of the aperture jitter is 20 ps, and the input signal is a 3-MHz sinusoid, for which values of the amplitude (in dBFS) will the aperture error power dominate the quantization noise power?

$\sigma_q^2 = \frac{\Delta^2}{12}.$
$\Delta = \frac{2 \cdot FS}{2^N}$
$\sigma_q^2 = \frac{1}{12}\left(\frac{2 \cdot FS}{2^N}\right)^2 = \frac{FS^2}{3 \cdot (2^N)^2}$
$\sigma_e^2 > \sigma_q^2$

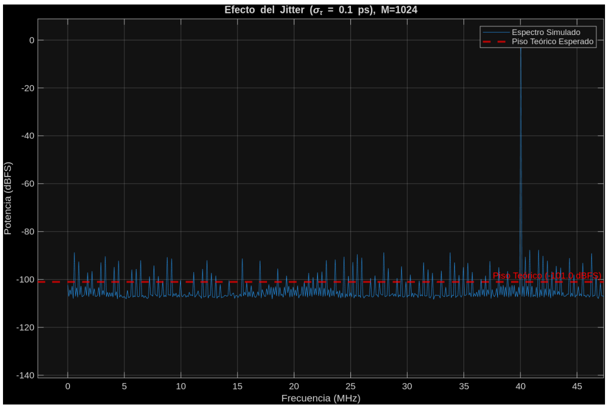
$$A_{dBFS} > 66{,}71 - 6{,}02 \cdot N$$

Question: Simulate the effect of aperture jitter on a full-scale sinusoid with frequency $40{,}03905$ MHz. Consider two cases: $\sigma_\tau = 10$ ps and $\sigma_\tau = 0{,}1$ ps respectively. Perform a 1024-FFT analysis of your data and check whether the perceived noise floor is at the expected level.

For $\sigma_\tau = 10ps$, the noise floor is-79.1 DBFS, and for $\sigma_\tau = 0,1ps$ is -101 DBFS



(a) $\sigma_\tau = 10ps$



(b) $\sigma_\tau = 0,1ps$

Question: Neglecting other possible sources of distortion, the total SNR is given by the ratio of the signal power to the sum of the powers of the noises due to jitter and quantization. Plot the theoretical total SNR (in dB) vs. input frequency over the range 0.1–100 MHz, assuming a full-scale sinusoid and for $\sigma_\tau \in \{10, 20, 40\}$ ps, $N \in \{10, 14\}$ bits (so that you should have six graphs in a single plot, whose x-axis should be in log scale). Comment on your results.

xd

A Appendix: MATLAB scripts and data

A.1 Task 3

A.1.1. task3_2.m

```
rng(0)

x0 = 1;
A = -x0; B = 0; C = +x0;

pd = makedist('Triangular','A',A,'B',B,'C',C);
N = 100000;
samples = random(pd, N, 1);

emp_mean = mean(samples);
```

```
emp_var = var(samples);
emp_desv_std = std(samples);

theo_var = (A^2 + B^2 + C^2 - A*B - A*C - B*C)/18;

rms = 20*log10(sqrt(emp_var)/x0);

fprintf('Theorical mean: 0; emp mean: %g\n',emp_mean);
fprintf('Theorical var: %.2f; emp var: %.2f\n',theo_var,emp_var);
fprintf('Sigma value: %.2f\n',sqrt(theo_var));
fprintf('rms value in dBFS: %g\n',rms)

xgrid = linspace(A,C,400)';
figure
histogram(samples,100,'Normalization','pdf', 'DisplayName', 'Generated
    samples')
hold on; grid on;

plot(xgrid, pdf(pd,xgrid), 'r-', 'LineWidth', 2, 'DisplayName', '
    Theoretical PDF');

title('Symmetric Triangular Distribution - makedist');
xlabel('Value');
ylabel('PDF');
legend('Location','best');
hold off
```

A.1.2. task3_tri.m

```
rng(0);

x0 = 1;
sigma_x = x0 / sqrt(6);
N = 100000;

a = x0 / 2;

x1 = (2*rand(N,1) - 1) * a;
x2 = (2*rand(N,1) - 1) * a;
y = x1 + x2;

mean_y = mean(y);
var_y = var(y,1);
rms_dBFS = 20*log10(sqrt(var_y)/x0);

fprintf('--- Validation ---\n');
fprintf('Expected mean = 0\n');
fprintf('Sample mean   = %.5f\n\n', mean_y);

fprintf('Expected RMS [dBFS] = -7.78\n');
fprintf('Sample RMS   [dBFS] = %g\n', rms_dBFS);

figure;
histogram(y, 100, 'Normalization', 'pdf', 'DisplayName', 'Generated
    samples');
hold on; grid on;
```

```
x_pdf = linspace(-x0, x0, 400);
f_pdf = (x0 - abs(x_pdf)) / (x0^2);
plot(x_pdf, f_pdf, 'r-', 'LineWidth', 2, 'DisplayName', 'Theoretical
    PDF');

title('Symmetric Triangular Distribution - Sum 2 dist');
xlabel('Value');
ylabel('PDF');
legend('Location','best');
hold off;
```

A.1.3. task3_normal_dist_set.m

```
rng(0);

x0 = 1;
sigma_x = 10^(-9.54/20);
N = 100000;

x = sigma_x * randn(1, N);

emp_mean = mean(x);
emp_var  = var(x,1);
rms_dBFS = 20*log10(sqrt(emp_var)/x0);

fprintf('--- Validation ---\n');
fprintf('Expected mean = 0\n');
fprintf('Sample mean   = %g\n\n', emp_mean);

fprintf('Expected RMS [dBFS] = -9.54\n');
fprintf('Sample RMS   [dBFS] = %g\n', rms_dBFS);

figure;
histogram(x, 60, 'Normalization', 'pdf');
hold on; grid on;

xx = linspace(-4*sigma_x, 4*sigma_x, 400);
plot(xx, (1/(sigma_x*sqrt(2*pi))) * exp(-0.5*(xx/sigma_x).^2), 'r-', '
    LineWidth',1.5);

title('Gaussian Distribution');
xlabel('Value');
ylabel('PDF');
legend('Empirical PDF','Theoretical');
grid on; hold off;
```

A.2 Task 4

A.2.1. task4_1.m

```
f0 = 37.1094e6;
M = 1024;
fs = 100e6;
Nbits = 12;
```

```
blocks = 15;
FS = 1;

Nsamples = blocks * M;

%% Generar FS sinusoidal
n = 0:Nsamples-1;
xt = FS * cos(2*n*pi*f0/fs);
xq = quanti(xt, FS, Nbits);

xqblocks = reshape(xq, M, 15);

X = fft(xqblocks, M);

P_avg = mean(abs(X).^2, 2);

norm_const = (M / 2)^2;
P_dbfs = 10 * log10(P_avg / norm_const);

f_axis = (0:M-1) * fs / M;
plot(f_axis(1:M/2 + 1) / 1e6, P_dbfs(1:M/2 + 1));
grid on;
xlabel('Frequency (MHz)');
ylabel('Power (dBFS)');
title('Espectro de Potencia Promedio (N=12 bits, M=1024)');
ylim([-140, 10]);
```

A.2.2. task4_2.m

```
f0 = 37.1094e6;
M = 256;
fs = 100e6;
Nbits = 12;
blocks = 15;
FS = 1;

Nsamples = blocks * M;

%% Generar FS sinusoidal
n = 0:Nsamples-1;
xt = FS * cos(2*n*pi*f0/fs);
xq = quanti(xt, FS, Nbits);

xqblocks = reshape(xq, M, 15);

X = fft(xqblocks, M);

P_avg = mean(abs(X).^2, 2);

norm_const = (M / 2)^2;
P_dbfs = 10 * log10(P_avg / norm_const);

f_axis = (0:M-1) * fs / M;
plot(f_axis(1:M/2 + 1) / 1e6, P_dbfs(1:M/2 + 1));
grid on;
xlabel('Frequency (MHz)');
```

```
ylabel('Power (dBFS)');
title('Espectro de Potencia Promedio (N=12 bits, M=256)');
ylim([-140, 10]);
```

A.2.3. task4_3.m

```
f0 = 37.1094e6;
M = 1024;
fs = 100e6;
blocks = 15;
FS = 1;
Nsamples = blocks * M;
f_axis = (0:M-1) * fs / M;
norm_const = (M / 2)^2;

n = 0:Nsamples-1;
xt = FS * cos(2*n*pi*f0/fs);

Nbits_list = [10, 8, 6];

for i = 1:length(Nbits_list)
    Nbits = Nbits_list(i);

    xq = quanti(xt, FS, Nbits);

    xqblocks = reshape(xq, M, blocks);
    X = fft(xqblocks, M);

    P_avg = mean(abs(X).^2, 2);
    P_dbfs = 10 * log10(P_avg / norm_const);

    sqnr_teorico = 6.02 * Nbits + 1.76;
    piso_ruido_teorico = -sqnr_teorico - 10*log10(M/2);

    figure;

    plot(f_axis(1:M/2 + 1) / 1e6, P_dbfs(1:M/2 + 1), 'b');
    hold on;
    yline(piso_ruido_teorico, 'r--', 'LineWidth', 1.5);
    hold off;

    grid on;
    xlabel('Frequency (MHz)');
    ylabel('Power (dBFS)');
    title(sprintf('Espectro (N = %d bits, M = 1024)', Nbits));
    ylim([-140, 10]);
    legend('Espectro medido', ...
           sprintf('Noise Floor (%.2f dBFS)', piso_ruido_teorico));
end
```

A.2.4. task4_4.m

```
f0 = 37.1094e6;
M = 1024;
fs = 100e6;
```

```
Nbits = 12;
blocks = 15;
FS = 1;

Nsamples = blocks * M;

n = 0:Nsamples-1;
xt = (1/3)*FS * cos(2*n*pi*f0/fs);
xq = quanti(xt, FS, Nbits);

xqblocks = reshape(xq, M, 15);

X = fft(xqblocks, M);

P_avg = mean(abs(X).^2, 2);

norm_const = (M / 2)^2;
P_dbfs = 10 * log10(P_avg / norm_const);

f_axis = (0:M-1) * fs / M;
plot(f_axis(1:M/2 + 1) / 1e6, P_dbfs(1:M/2 + 1));
grid on;
xlabel('Frequency (MHz)');
ylabel('Power (dBFS)');
title('Espectro de Potencia Promedio (N=12 bits, M=1024)');
ylim([-140, 10]);
```

A.2.5. task4_5.m

```
f0 = 37.12e6;
M = 1024;
fs = 100e6;
Nbits = 16;
%blocks = 15;
blocks = 100;
FS = 1;

Nsamples = blocks * M;

n = 0:Nsamples-1;
xt = FS * cos(2*n*pi*f0/fs);
xq = quanti(xt, FS, Nbits);

xqblocks = reshape(xq, M, blocks);

X = fft(xqblocks, M);

P_avg = mean(abs(X).^2, 2);

norm_const = (M / 2)^2;
P_dbfs = 10 * log10(P_avg / norm_const);

f_axis = (0:M-1) * fs / M;
plot(f_axis(1:M/2 + 1) / 1e6, P_dbfs(1:M/2 + 1));
grid on;
```

```
xlabel('Frequency (MHz)');
ylabel('Power (dBFS)');
title('Espectro de Potencia Promedio (N=16 bits, M=1024)');
ylim([-140, 10]);
```

A.3 Task 5

A.3.1. task5_1.m

```
FS = 1;
x = linspace(-FS, FS, 1000);

g_0 = x;

gama_1 = 1;
g_1 = sign(x) .* (FS / log(1 + gama_1)) .* log(1 + gama_1 .* abs(x) /
    FS);
g_1(x == 0) = 0;

gama_2 = 2;
g_2 = sign(x) .* (FS / log(1 + gama_2)) .* log(1 + gama_2 .* abs(x) /
    FS);
g_2(x == 0) = 0;

figure;
plot(x, g_0, 'b', 'LineWidth', 2);
hold on;
plot(x, g_1, 'r', 'LineWidth', 2);
plot(x, g_2, 'g', 'LineWidth', 2);
grid on;
xlabel('x');
ylabel('g(x)');
title('Distortion Function g_\gamma(x)');
legend('\gamma = 0 (Ideal)', '\gamma = 1', '\gamma = 2');
```

A.3.2. task5_3.m

A.3.3. task5_4.m

```
f0 = 6.8359e6;
fs = 100e6;
FS = 1;
gamma_list = [0.01,0.1];
Nbits = 11;
M = 2048;
blocks = 15;

for gamma = gamma_list

    Nsamples = blocks * M;
    n = (0:Nsamples-1).';
    xt = FS * cos(2*pi*f0/fs * n);
```

```
    xq = dquanti(xt, FS, Nbits, gamma);
    xqblocks = reshape(xq, M, blocks);

    X = fft(xqblocks, M);

    P_avg = mean(abs(X).^2, 2);

    k0 = round(f0 * M / fs);
    n0 = (0:M-1).';
    xref = FS * cos(2*pi*(k0/M) * n0);
    Pref = max(abs(fft(xref, M)).^2);

    half = 1:(M/2);
    freqs = (half-1) * (fs / M);
    P_half = P_avg(half);

    P_dbfs = 10*log10( P_half / Pref );

    figure('Name',sprintf('M=%d, y=%.2f',M,gamma));
    plot(freqs/1e6, P_dbfs, 'LineWidth', 1.2);
    title(sprintf('PSD averaged, M=%d, N=%d, \\gamma=%.4g', M, Nbits,
   gamma));
    legend('PSD (avg)');
    grid on;
end
```

A.3.4. task5_5.m

```
f0 = 6.8359e6;
fs = 100e6;
FS = 1;
gamma_list = [0.005,0.05,0.1];
Nbits = 11;
M_list = [2048, 512];
blocks = 15;

for M = M_list
    for gamma = gamma_list
        Nsamples = blocks * M;
        n = (0:Nsamples-1).';
        xt = (FS/3) * cos(2*pi*f0/fs * n);

        xq = dquanti(xt, FS, Nbits, gamma);
        xqblocks = reshape(xq, M, blocks);

        X = fft(xqblocks, M);

        P_avg = mean(abs(X).^2, 2);

        k0 = round(f0 * M / fs);
        n0 = (0:M-1).';
        xref = FS * cos(2*pi*(k0/M) * n0);
        Pref = max(abs(fft(xref, M)).^2);

        half = 1:(M/2);
        freqs = (half-1) * (fs / M);
```

```
        P_half = P_avg(half);

        P_dbfs = 10*log10( P_half / Pref );

        figure('Name',sprintf('M=%d, y=%.2f',M,gamma));
        plot(freqs/1e6, P_dbfs, 'LineWidth', 1.2);
        title(sprintf('PSD averaged, M=%d, N=%d, \\gamma=%.4g', M,
   Nbits, gamma));
        legend('PSD (avg)');
        grid on;
    end
end
```

A.3.5. task5_6.m

```
f0 = 3.3202e6;
fs = 100e6;
FS = 1;
gamma = 0.3;
Nbits = 11;
M = 2048;
blocks = 15;

Nsamples = blocks * M;
n = (0:Nsamples-1).';
xt = (FS/2) * cos(2*pi*f0/fs * n);

xq = dquanti(xt, FS, Nbits, gamma);
xqblocks = reshape(xq, M, blocks);

X = fft(xqblocks, M);

P_avg = mean(abs(X).^2, 2);

k0 = round(f0 * M / fs);
n0 = (0:M-1).';
xref = FS * cos(2*pi*(k0/M) * n0);
Pref = max(abs(fft(xref, M)).^2);

half = 1:(M/2);
freqs = (half-1) * (fs / M);
P_half = P_avg(half);

P_dbfs = 10*log10( P_half / Pref );

figure('Name',sprintf('M=%d, y=%.2f',M,gamma));
plot(freqs/1e6, P_dbfs, 'LineWidth', 1.2);
title(sprintf('PSD averaged, M=%d, N=%d, \\gamma=%.4g', M, Nbits, gamma
   ));
legend('PSD (avg)');
grid on;
```

A.4 Task 6

A.4.1. task6_3.m

```
fs = 100e6;
Nbits = 12;
FS = 1;
M = 1024;
blocks = 100;
Nsamples = M * blocks;

k0 = 410;
fc = k0 * fs / M;


sigma_list_ps = [10, 0.1];

f_axis = (0:M/2) * fs / M;

Pq_dBFS = -(6.02 * Nbits + 1.76);
Pq_linear = 10^(Pq_dBFS / 10);
FFT_gain_dB = 10 * log10(M / 2);

for sigma_ps = sigma_list_ps
    sigma_tau = sigma_ps * 1e-12;

    SNR_jitter_dB = 20 * log10(1 / (2 * pi * fc * sigma_tau));
    Pj_dBFS = -SNR_jitter_dB;
    Pj_linear = 10^(Pj_dBFS / 10);

    P_total_linear = Pq_linear + Pj_linear;
    P_total_dBFS = 10 * log10(P_total_linear);

    Expected_Floor_dBFS = P_total_dBFS - FFT_gain_dB;

    fprintf('--- Caso sigma = %.1f ps ---\n', sigma_ps);
    fprintf('  P_cuantizacion (Pq): %.2f dBFS\n', Pq_dBFS);
    fprintf('  P_jitter (Pj):       %.2f dBFS\n', Pj_dBFS);
    fprintf('  P_ruido_total:       %.2f dBFS\n', P_total_dBFS);
    fprintf('  Piso FFT Esperado:   %.2f dBFS\n', Expected_Floor_dBFS);

    if sigma_ps == 10
        floor_10ps = Expected_Floor_dBFS;
    else
        floor_0_1ps = Expected_Floor_dBFS;
    end
end


for sigma_ps = sigma_list_ps
    sigma_tau = sigma_ps * 1e-12;

    n = (0:Nsamples-1)';
    t_ideal = n / fs;

    a = sigma_tau * sqrt(3);
    tau_n = -a + (2 * a) * rand(Nsamples, 1);

    t_jittered = t_ideal + tau_n;

    xt = FS * cos(2 * pi * fc * t_jittered);
```

```
    xq = quanti(xt, FS, Nbits);

    xq_blocks = reshape(xq, M, blocks);

    X_fft = fft(xq_blocks, M);
    P_avg = mean(abs(X_fft).^2, 2);

    norm_const = (M / 2)^2;
    P_dbfs = 10 * log10(P_avg / norm_const);

    figure;
    plot(f_axis / 1e6, P_dbfs(1:M/2 + 1));
    hold on;

    if sigma_ps == 10
        yline(floor_10ps, 'r--', 'LineWidth', 2, ...
            'Label', sprintf('Piso Teorico (%.1f dBFS)', floor_10ps));
    else
        yline(floor_0_1ps, 'r--', 'LineWidth', 2, ...
            'Label', sprintf('Piso Teorico (%.1f dBFS)', floor_0_1ps));
    end

    grid on;
    title(sprintf('Efecto del Jitter (\\sigma_{\\tau} = %.1f ps), M
   =1024', sigma_ps));
    xlabel('Frecuencia (MHz)');
    ylabel('Potencia (dBFS)');
    ylim([-140, 10]);
    legend('Espectro Simulado', 'Piso Teorico Esperado');
end
```