5.5pts

Procesamiento de Señales, Fundamentos

Trabajo Práctico 1

Jairo Mena

Profesor:

Pablo Slavkin

Universidad de Buenos Aires

MSE 5Co2020

1) Demostración que las siguientes ecuaciones cumplen con las propiedades de linealidad e invariación en el tiempo para determinar si con sistemas LTI:

a) y(t) = x(t)∗cos(t)

Para **x(t)** se realiza desplazamiento temporal
Si **$x(t-t_0)$** dá como resultado**: $x(t-t_0)$*cos(t)**
Para **y(t)** se realiza desplazamiento temporal
Si **$y(t-t_0)$** dá como resultado: **$x(t-t_0)$*cos(t- t_0)**

$$x(t-t_0)*\cos(t) \neq x(t-t_0)*\cos(t-t_0)$$

Lo anterior demuestra que **y(t) = x(t)∗cos(t)** no es un sistema invariante en el tiempo, por lo tanto, no es un sistema LTI.

b) y(t) = cos(x(t))

Para **x(t)** se aplica propiedad de proporcionalidad
Si **αx(t)** dá como resultado **cos(αx(t))**
Para **y(t)** se aplica propiedad de proporcionalidad
Si **αy(t)** dá como resultado **αcos(x(t))**

$$\cos(\alpha x(t)) \neq \alpha\cos(x(t))$$

Lo anterior demuestra que **y(t) = cos(x(t))** no cumple con la propiedad de linealidad de proporcionalidad, por lo tanto, no es un sistema LTI.

c) y(t) =$e^{x(t)}$

Para **x(t)** se aplica propiedad de proporcionalidad
Si **αx(t)** dá como resultado **$e^{\alpha x(t)}$**
Para **y(t)** se aplica propiedad de proporcionalidad
Si **αy(t)** dá como resultado **$\alpha e^{x(t)}$**

$$e^{\alpha x(t)} \neq \alpha e^{x(t)}$$

Lo anterior demuestra que **y(t) =$e^{x(t)}$** no cumple con la propiedad de linealidad de proporcionalidad, por lo tanto, no es un sistema LTI.

d) $y(t) = \frac{1}{2} x(t)$

Demostración de las propiedades de linealidad:

Para **x(t)** se aplica propiedad de proporcionalidad.
Si $\alpha x(t)$ dá como resultado $\frac{1}{2}\alpha x(t)$
Para **y(t)** se aplica propiedad de proporcionalidad.
Si $\alpha y(t)$ dá como resultado $\alpha \frac{1}{2} x(t)$

$$\frac{1}{2}\alpha x(t) = \alpha \frac{1}{2} x(t)$$

Para $x_1(t)$ y $x_2(t)$ se aplica propiedad aditivita.
Si $x_1(t) + x_2(t)$ entonces debe cumplir $y_1(t) + y_2(t)$

$$y_1(t) + y_2(t) = \frac{1}{2}(x_1(t) + x_2(t)) = \frac{1}{2} x_1(t) + \frac{1}{2} x_1(t)$$

Para $x_1(t)$ y $x_2(t)$ se aplica propiedad de superposición proporcional.
Si $\alpha x_1(t) + \beta x_2(t)$ entonces debe cumplir $\alpha y_1(t) + \beta y_2(t)$

$$\alpha y_1(t) + \beta y_2(t) = \frac{1}{2}(\alpha x_1(t) + \beta x_2(t)) = \frac{1}{2}\alpha x_1(t) + \frac{1}{2}\beta x_1(t)$$

Para **x(t)** se realiza desplazamiento temporal.
Si $x(t-t_0)$ dá como resultado: $\frac{1}{2} x(t-t_0)$
Para **y(t)** se realiza desplazamiento temporal.
Si $y(t-t_0)$ dá como resultado: $\frac{1}{2} x(t-t_0)$

$$\frac{1}{2} x(t-t_0) = \frac{1}{2} x(t-t_0)$$

Lo anterior demuestra que $y(t) = \frac{1}{2} x(t)$ cumple con todas las propiedades de linealidad y es un sistema invariante en el tiempo, por lo tanto, es un sistema LTI.

2) Ruido de cuantización:

- 24 bits

  SNR = 1,76 + 6,02 ∗ **24** = 146,24dB

- 16 bits

  SNR = 1,76 + 6,02 ∗ **16** = 98,08dB

- 10 bits

  SNR = 1,76 + 6,02 ∗**10** = 61,96dB

- 8 bits

  SNR = 1,76 + 6,02 ∗ **8** = 49,92dB

- 2 bits

  SNR = 1,76 + 6,02∗ **2** = 13,8dB

3) Utilizo la técnica de sobre muestreo, que consiste en realizar el muestreo a una frecuencia mayor de la que se está trabajando, de esta forma se hace que la densidad espectral sea menor. Lo anterior teniendo en cuenta, siempre y cuando el ruido no sea inherente a la señal de entrada.

$$S_{espectral} = \frac{P_q}{Fs}$$

Como se aprecia en la ecuación de densidad espectral aplicando una frecuencia de muestreo cuatro veces la frecuencia de muestreo anterior se obtiene 6dB de extras en la relación señal a ruido, utilizando los mismos 10 bits de SNR.

Aplicando sobre muestreo se obtiene:

$$S_{espectral} = \frac{P_q}{4\,Fs}$$

4) Filtro antialias del trabajo final:

Para el trabajo final que consiste en la medición de los armónicos de las variables de corriente, voltaje y potencia, se va a medir dos entradas:

- Señal proveniente de un reductor de voltaje (Divisor) que está conectado al voltaje AC de 110V@60Hz.
- Señal proveniente de un transformador de corriente, que mide la corriente que pasa por un conductor.

Para las dos señales se requiere que la banda de interés es de (40Hz a 5kHz), ya que es el valor recomendable en donde se encuentran los posibles armónicos de las señales, sin embargo, para que no se admitan señales no deseadas por la caída de la curva del filtro de primer orden, se realiza el diseño del filtro con una frecuencia de corte menor a los de 5kHz y se escoge por facilidad del diseño 4.8kHz.

La frecuencia de muestreo del sistema será de 10kHz y se utiliza el mismo filtro para los dos canales.

Se utilizará un filtro pasa bajo de primer orden.

Por lo tanto si, R = 1kΩ y f_{corte} = 4.8kHz.

$$\boldsymbol{C = \frac{1}{2\pi R f_{corte}}}$$

$$\boldsymbol{C = \frac{1}{2\pi(1k\Omega)(4,8kHz)}}$$

$$\boldsymbol{C \approx 33.15nF}$$

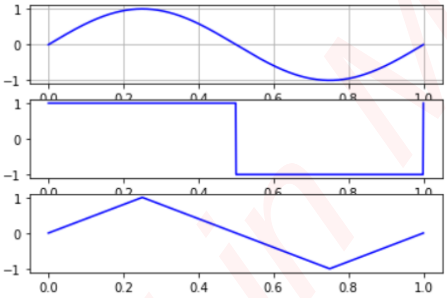
Se escoge un valor de 33nF ya que es el valor que comercialmente se consigue.

Generación y simulación

1. Módulo o paquete con las siguientes funciones

- senoidal (fs[Hz], f0[Hz], amp[0 a 1], muestras),fase [radianes]
- Cuadrada (fs[Hz], f0[Hz], amp[0 a 1], muestras)
- Triangular(fs[Hz], f0[Hz], amp[0 a 1], muestras)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as scp

fig = plt.figure(10)

fs = 1000
f0 = 1
Amp = 1
faseSen = 0 * np.pi
faseSq = 0 * np.pi
faseSaw = 0.5 * np.pi

N = np.linspace(0,f0,fs)

sen = fig.add_subplot(3,1,1)
sen.plot(N, Amp * np.sin((2*np.pi*N + faseSen)),"b-")

sq = fig.add_subplot(3,1,2)
sq.plot(N, Amp * scp.square((2 * np.pi * f0 * N) + faseSq, Amp/2),"b-")

saw = fig.add_subplot(3,1,3)
saw.plot(N, scp.sawtooth((2 * np.pi * f0 * N) + faseSaw, Amp/2),"b-")

sen.grid(True)
plt.show()
```

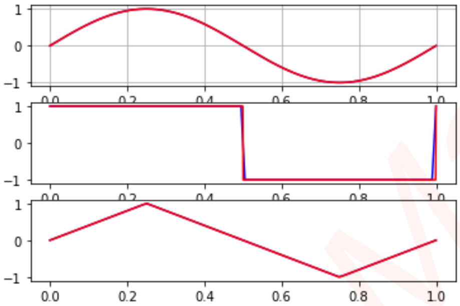


2. Se realiza los siguientes experimentos

Con señales con parámetros:

- fs = 1000
- N = 1000
- fase = 0
- amp = 1

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as scp

fig = plt.figure(10)

fs1 = 100
fs2 = 1100

f0 = 1
Amp = 1
faseSen = 0 * np.pi
faseSq = 0 * np.pi
faseSaw = 0.5 * np.pi

N1 = np.linspace(0,f0,fs1)
N2 = np.linspace(0,f0,fs2)

sen = fig.add_subplot(3,1,1)
sen.plot(N1, Amp * np.sin((2*np.pi*N1 + faseSen)),"b-")
sen.plot(N2, Amp * np.sin((2*np.pi*N2 + faseSen)),"r-")

sq = fig.add_subplot(3,1,2)
sq.plot(N1, Amp * scp.square((2 * np.pi * f0 * N1) + faseSq, Amp/2),"b-")
sq.plot(N2, Amp * scp.square((2 * np.pi * f0 * N2) + faseSq, Amp/2),"r-")

saw = fig.add_subplot(3,1,3)
saw.plot(N1, scp.sawtooth((2 * np.pi * f0 * N1) + faseSaw, Amp/2),"b-")
saw.plot(N2, scp.sawtooth((2 * np.pi * f0 * N2) + faseSaw, Amp/2),"r-")

sen.grid(True)
plt.show()
```

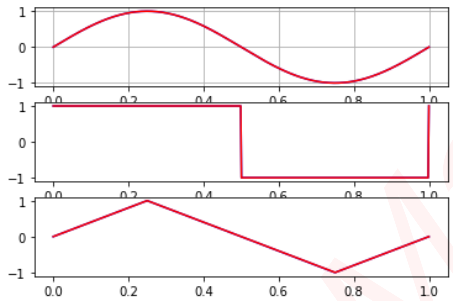


2.1. Se puede visualizar en la anterior gráfica que cuando se realiza un cambio así sea de un valor grande en la frecuencia de muestreo ***Fs***, los resultados son casi imperceptibles, sin embargo, cuando las señales tienen altas frecuencias, como en la gráfica de la señal cuadrada se puede notar que hay una pequeña diferencia.

Lo anterior nos demuestra en forma práctica la teoría de Nyquist, porque las altas frecuencias en los cambios abruptos de la señal cuadrada se encuentran por encima de la frecuencia de muestreo, por lo tanto, se puede visualizar las diferencias.

Las frecuencias de las otras señales se encuentran muy por debajo de la frecuencia de Nyquist, por lo tanto, no se percibe ninguna diferencia.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as scp

fig = plt.figure(10)

fs1 = 490
fs2 = 510

f0 = 1
Amp = 1
faseSen = 0 * np.pi
faseSq = 0 * np.pi
faseSaw = 0.5 * np.pi

N1 = np.linspace(0,f0,fs1)
N2 = np.linspace(0,f0,fs2)

sen = fig.add_subplot(3,1,1)
sen.plot(N1, Amp * np.sin((2*np.pi*N1 + faseSen)),"b-")
sen.plot(N2, Amp * np.sin((2*np.pi*N2 + faseSen)),"r-")

sq = fig.add_subplot(3,1,2)
sq.plot(N1, Amp * scp.square((2 * np.pi * f0 * N1) + faseSq, Amp/2),"b-")
sq.plot(N2, Amp * scp.square((2 * np.pi * f0 * N2) + faseSq, Amp/2),"r-")

saw = fig.add_subplot(3,1,3)
saw.plot(N1, scp.sawtooth((2 * np.pi * f0 * N1) + faseSaw, Amp/2),"b-")
saw.plot(N2, scp.sawtooth((2 * np.pi * f0 * N2) + faseSaw, Amp/2),"r-")

sen.grid(True)
plt.show()
```



2.2. En la gráfica anterior se puede vislumbrar que los cambios de la frecuencia de muestreo es muy pequeño, por lo tanto, los resultados se aproximan mucho a su igualdad, se nota en las gráficas que las dos señales se sobreponen una con otra. Lo anterior demuestra de nuevo el teorema de Nyquist.

En este caso las frecuencias de muestreo se acercan a 500 veces la señal, por lo tanto, no se encuentra diferencia.

Si bien el analisis que haces esta bien, no es lo que se pide en los puntos 2.1 y 2.2. Lo que se pide no es cambiar Fs y dejar F0 fija, sino que F0 cambia y vale una porcion de Fs. Para cualquier Fs, 2.1 pide que f0 valga un 10% de fs y otra f0 un 110%. Lo mismo para el 2.2
Estaria bueno si queres hacerlo porque es un ejercicion bastante revelador y que invita a pensar en varias cosas y mas ahora que manejas DFT. Las conclusiones seran sensiblemente diferentes a las que obtuvistes asi como tambien los graficos, sin embargo el codigo lo podes aprovechar al 99%.

Adquisición y reconstrucción

1. Se genera un tono de LA-440. Y se digitaliza con 10, 8, 4 y 2 bits con el ADC, se envía los datos al PC, y se muestra en pantalla.

1.1. Señal original con su máximo, mínimo y RMS.

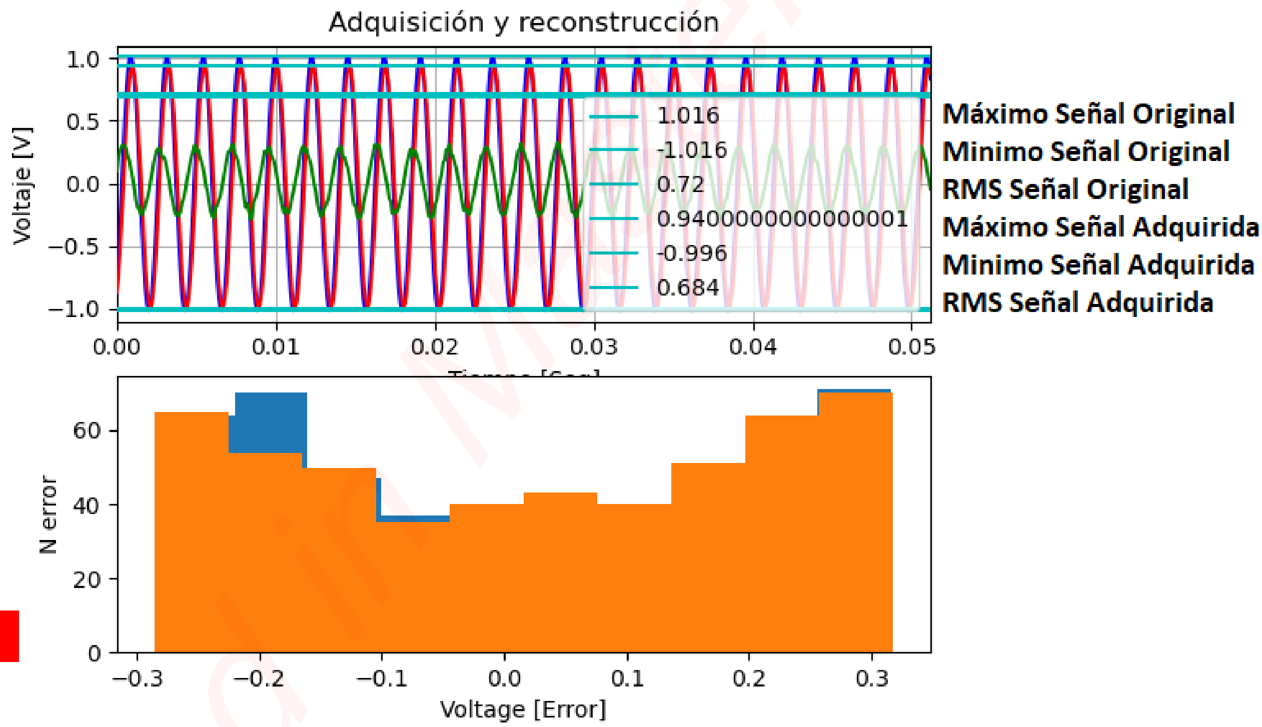
1.2. Señal adquirida con su máximo, mínimo y RMS.

1.3. Señal error = Original – Adquirida

1.4. Histograma de error

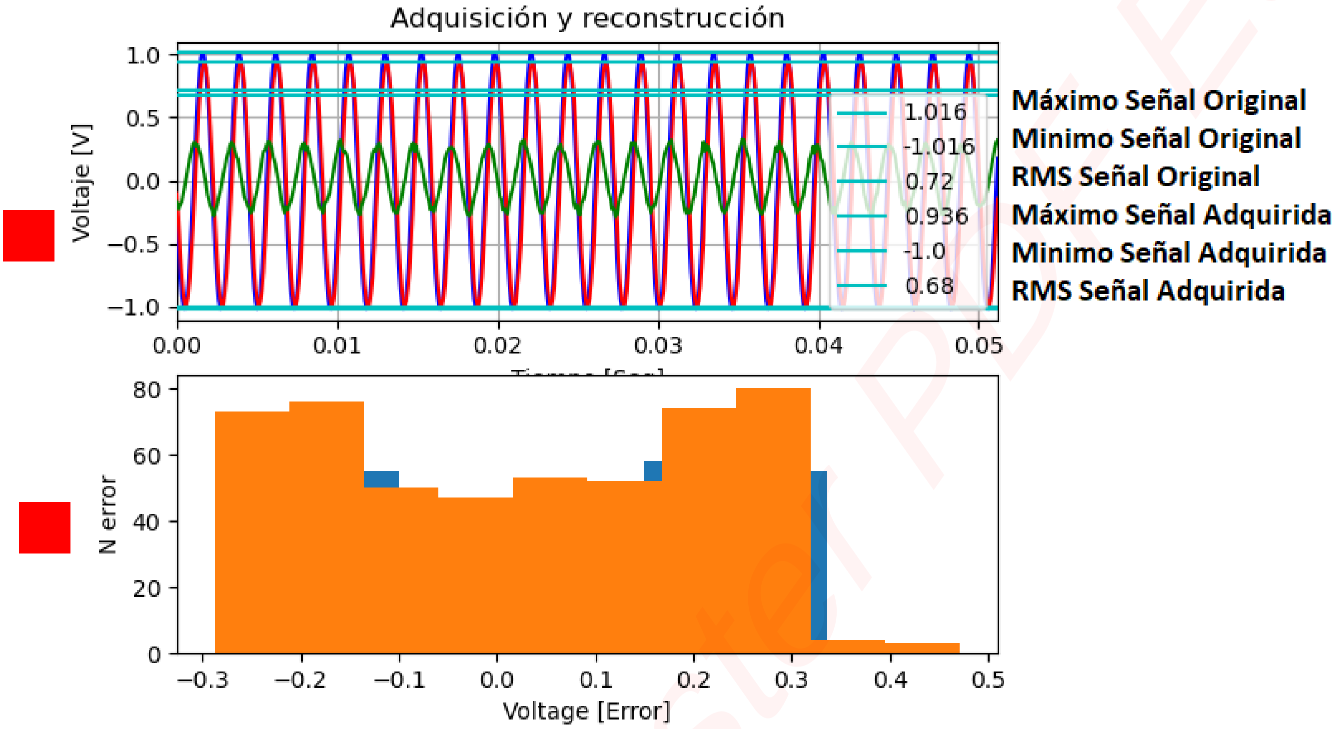
Señal tono LA-440 con digitalización con 10 bits

Porque hay tanta atenuacion entre las señales generada y adquirida?

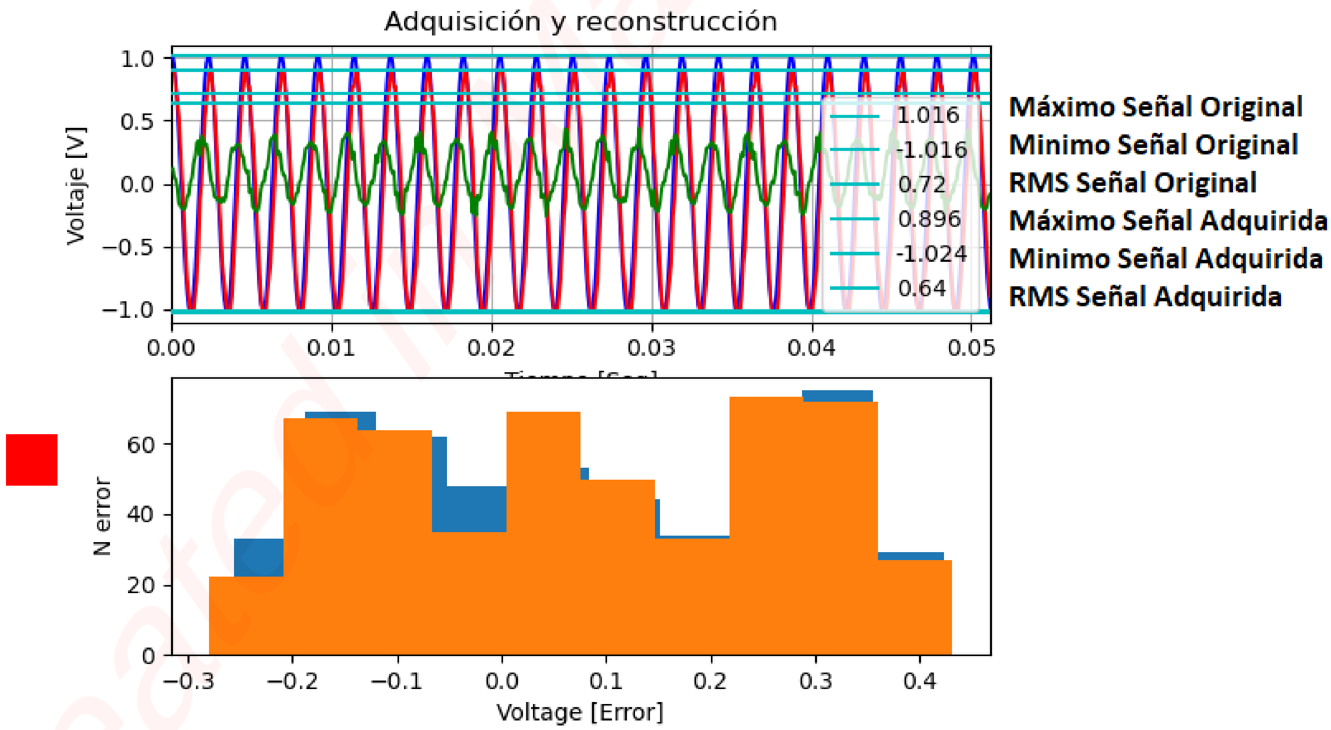


Porque el histograma tiene de fondo algo en azul? que representa?
Si la representacion en X esta en volts, habria que normarlizar la escala para conocer el maximo y el minimo. O bien cuantos volts representa 1 LSB

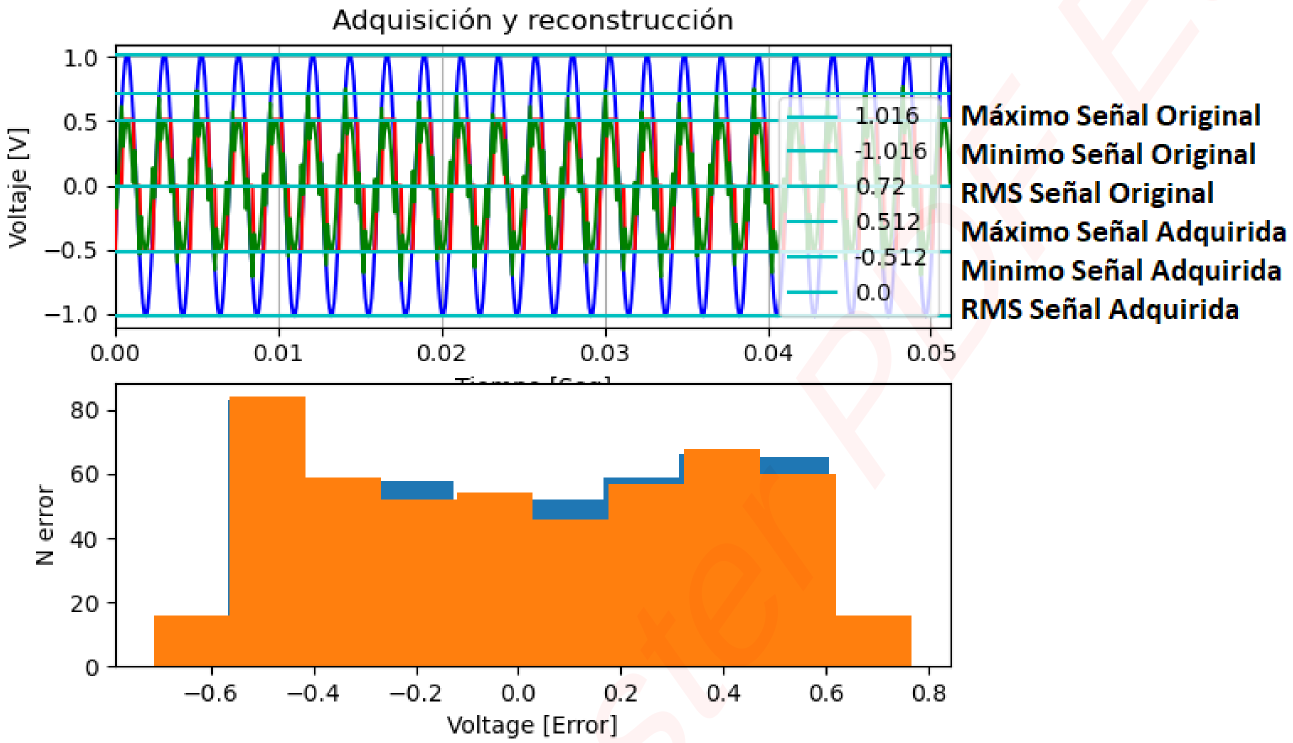
Señal tono LA-440 con digitalización con 8 bits



Señal tono LA-440 con digitalización con 4 bits



Señal tono LA-440 con digitalización con 2 bits



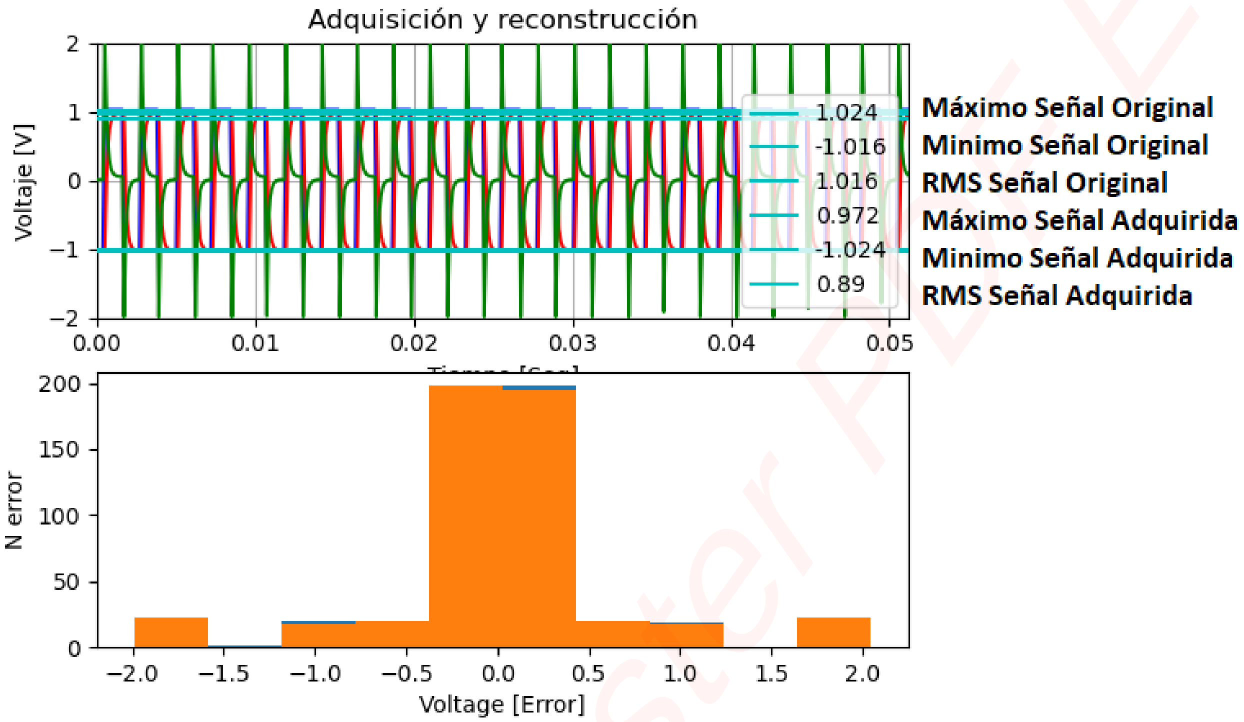
La distribución del error de cuantificación en las digitalización con un número de bits de 10 y 8 bits es muy parecida, tiende a tener más error a los extremos de los voltajes límites de conversión. Se puede concluir que lo anterior debido a que existe un pequeño desfase entre la señal original y la adquirida.

Sin embargo cuando el número de bits de digitalización es de 4 y 2 bits el error se distribuye de forma aproximada homogénea, se puede concluir que es porque se genera el error de cuantificación en toda la señal.

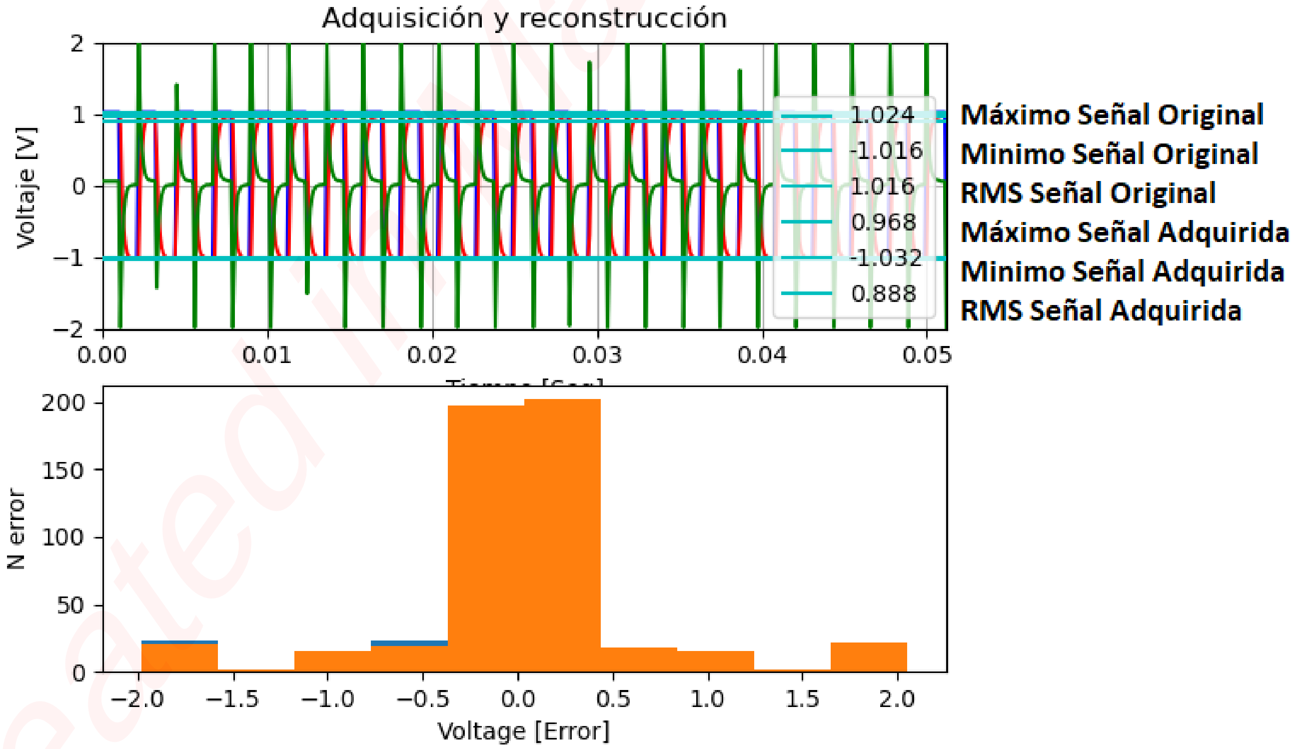
El histograma de error para una señal senoidal como vimos en clases es mas o menos homogenea ya que tarde o temprano la adquisicion del ADC comete un error que va de -1/2LSB a +1/2LSB de forma
Si en algun caso la señal estuviera desfasada de la genearad por un periodo de tiempo, eso no afectaria la distribucion. Lo que no esta correcto son las escalas del eje X y que haya 2 colores en el histograma como que hay 2 analisis superpuestos, pero se trata solo de analizar un histograma, el de la señal eror, la diferencia entre lo que el ADC debio tomar y lo que en realidad tomo.

No alcanzo a apreciar las curvas, te quedo muy amontonado, Pareciera que hay una curva verde que cae de modo exponencial. Te hubiera convenido mostrar las curvas en varios graficos o hacer un zoom en la parte de interes.

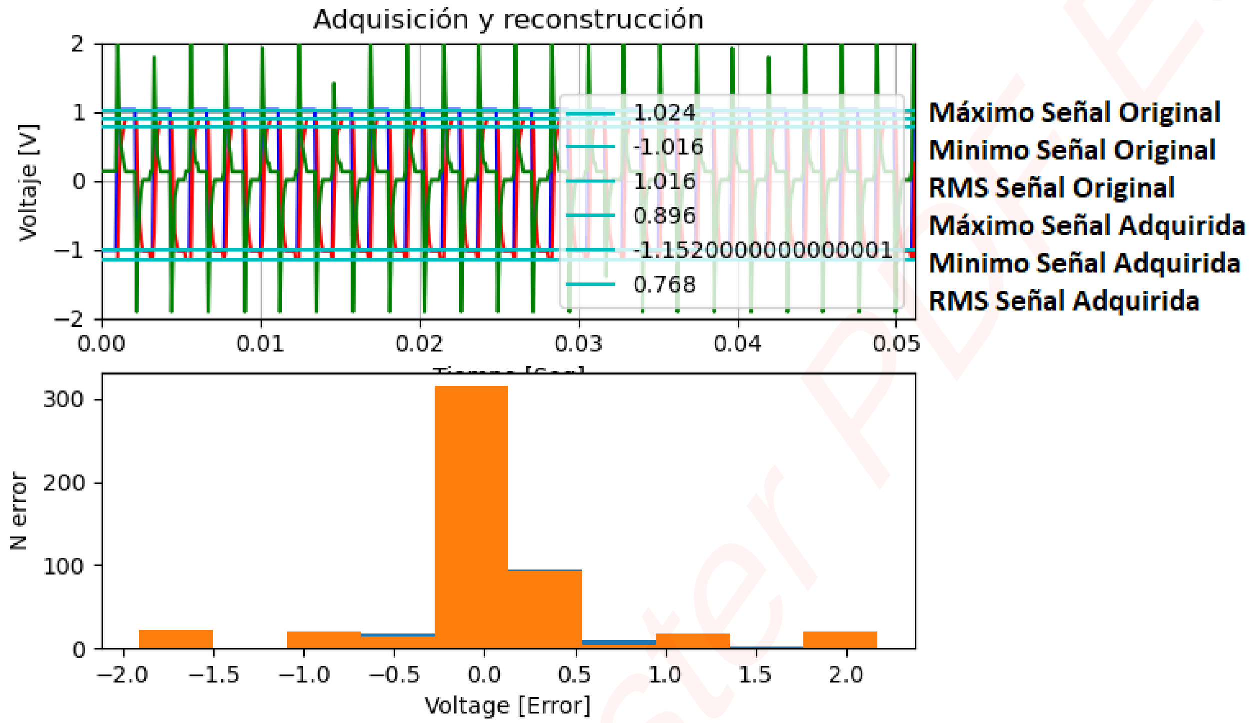
2. Señal Cuadrada con digitalización con 10 bits



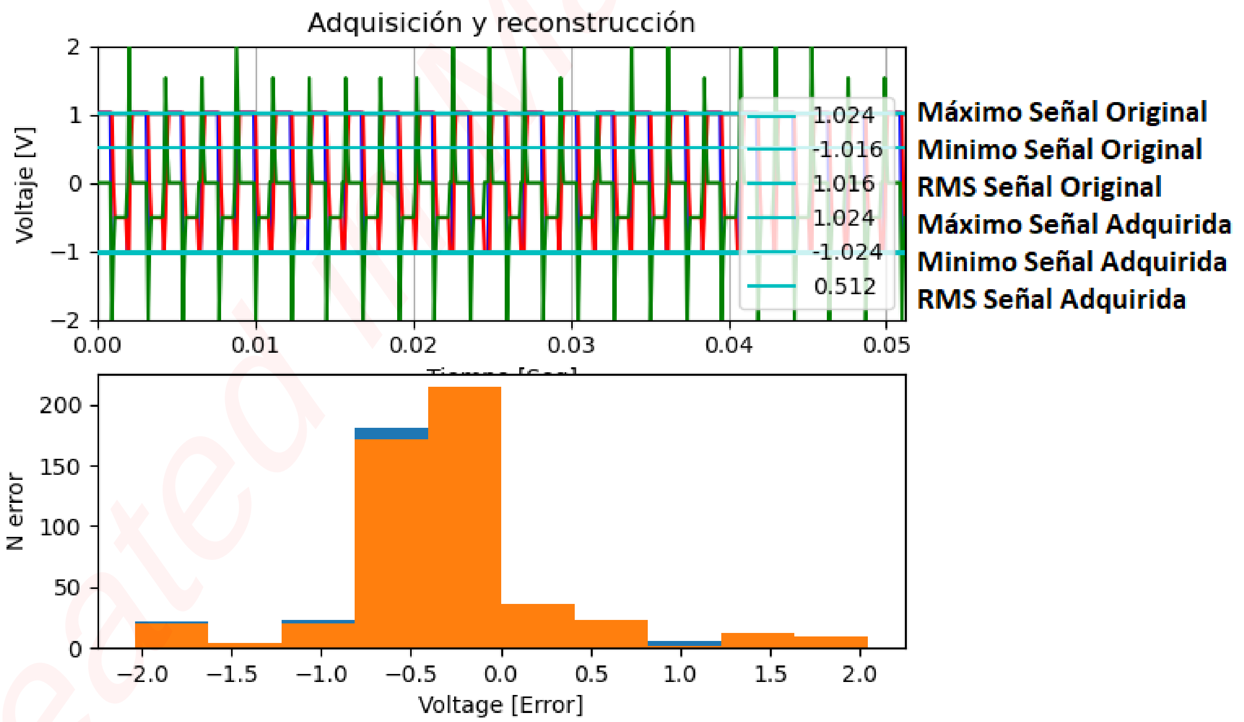
Señal Cuadrada con digitalización con 8 bits



Señal Cuadrada con digitalización con 4 bits



Señal Cuadrada con digitalización con 2 bits

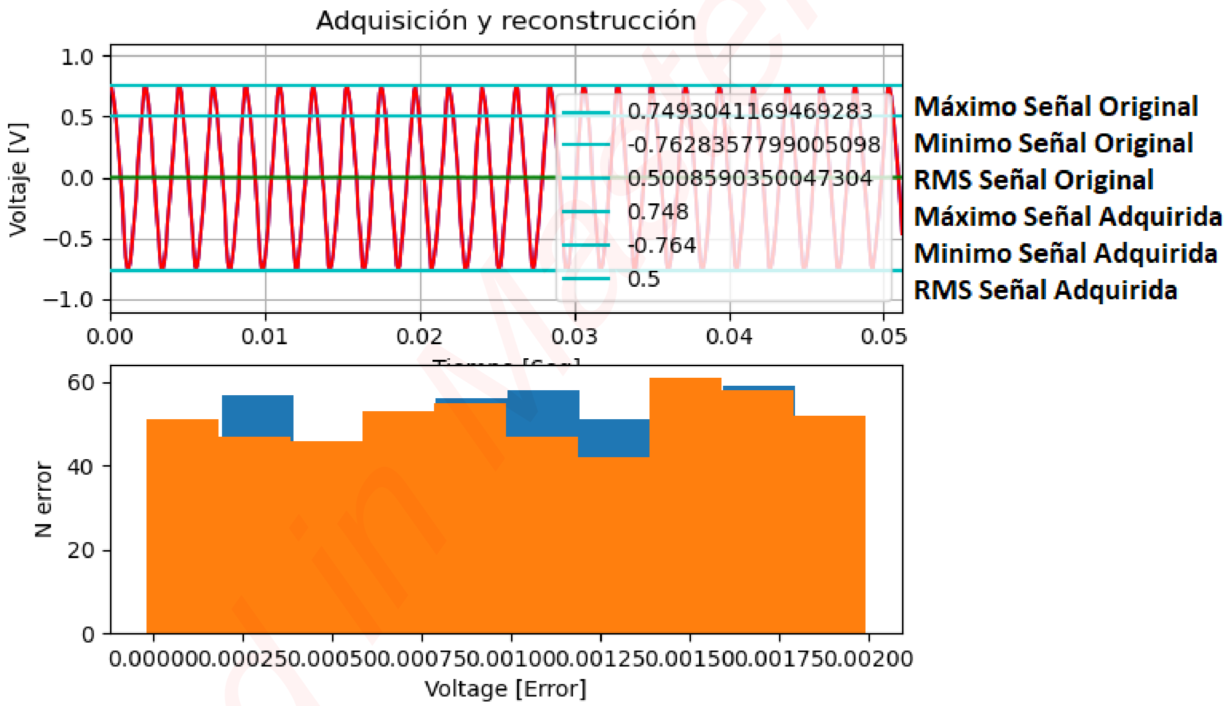


En la señal cuadrada se puede apreciar que el error se establece en la parte cercana a los cero voltios y es debido a que los máximos picos de la señal de la diferencia de las señales (señal de error) se encuentra en el cambio abrupto de cero al mayor voltaje.

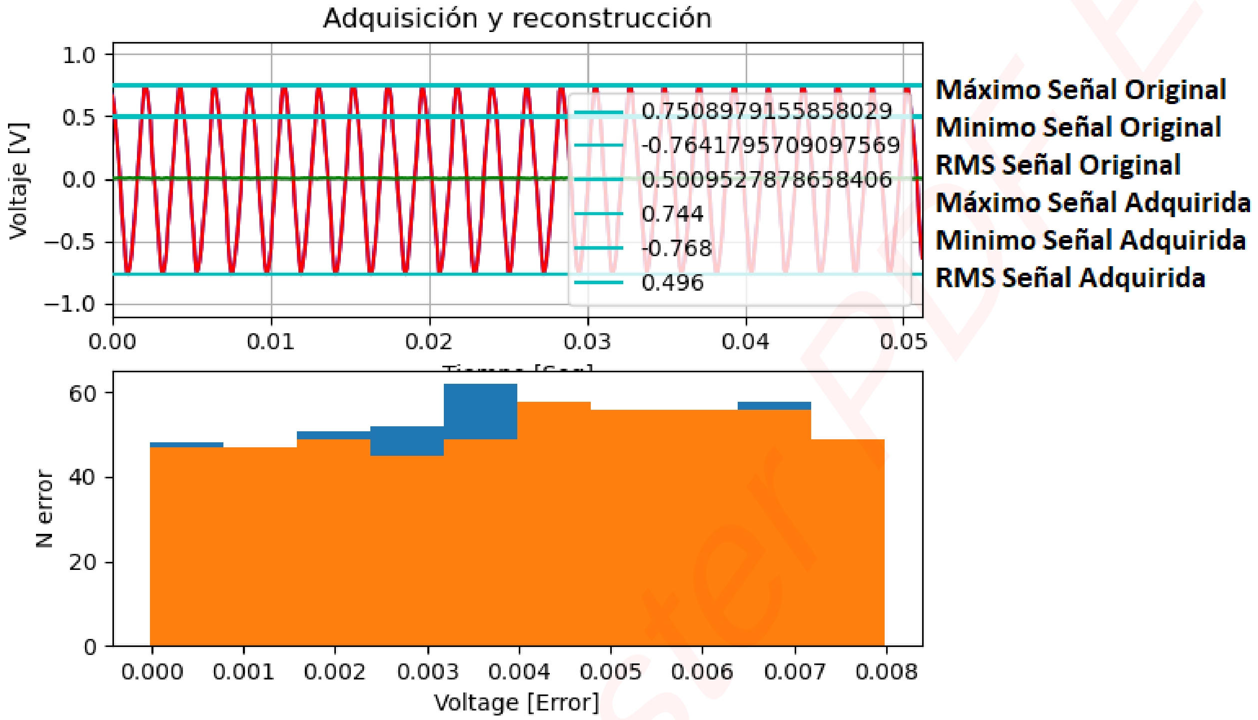
En todas las digitalizaciones se aprecia el mismo fenómeno. Y se puede concluir que como es una señal que tienen solamente dos amplitudes entonces no hay mucha diferencia entre los números de bits en las digitalizaciones.

La conclusion esperable aca es que al haber solo 2 amplitudes, el histograma de error se concentra en solo dos lugares, uno en la zona de error al capturar el estao alto y otra torre en la zona donde el ADC se equivoca leyendo el estado bajo.

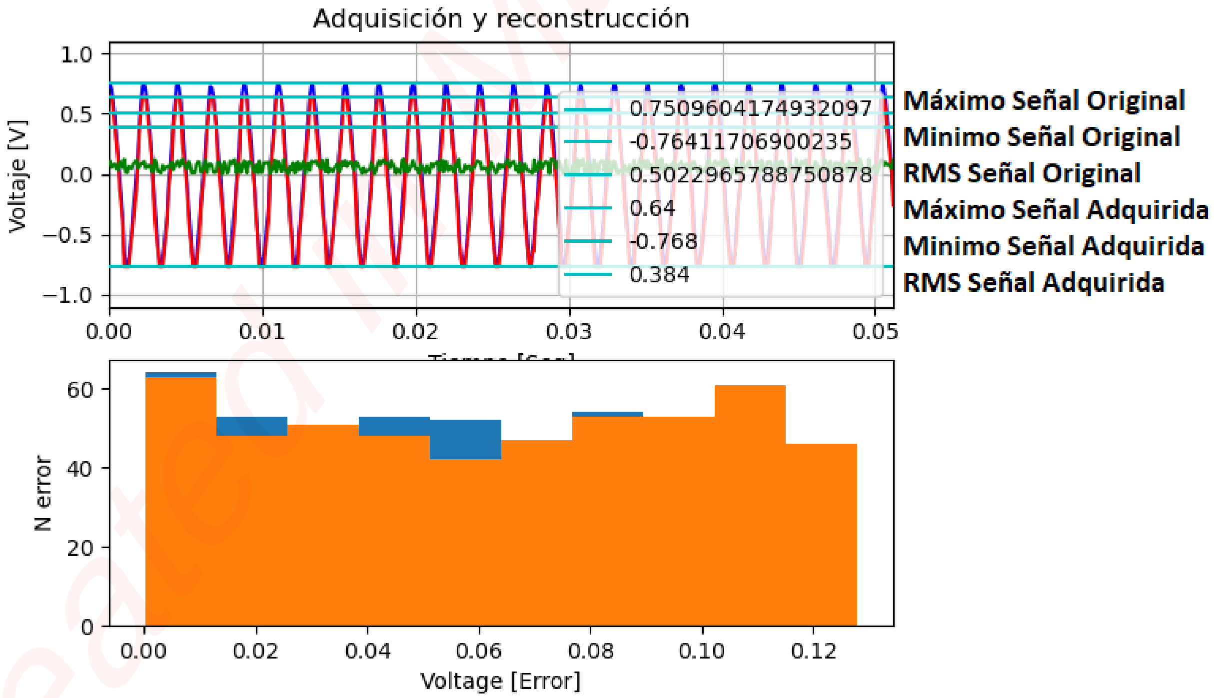
Señal Triangular con digitalización con 10 bits



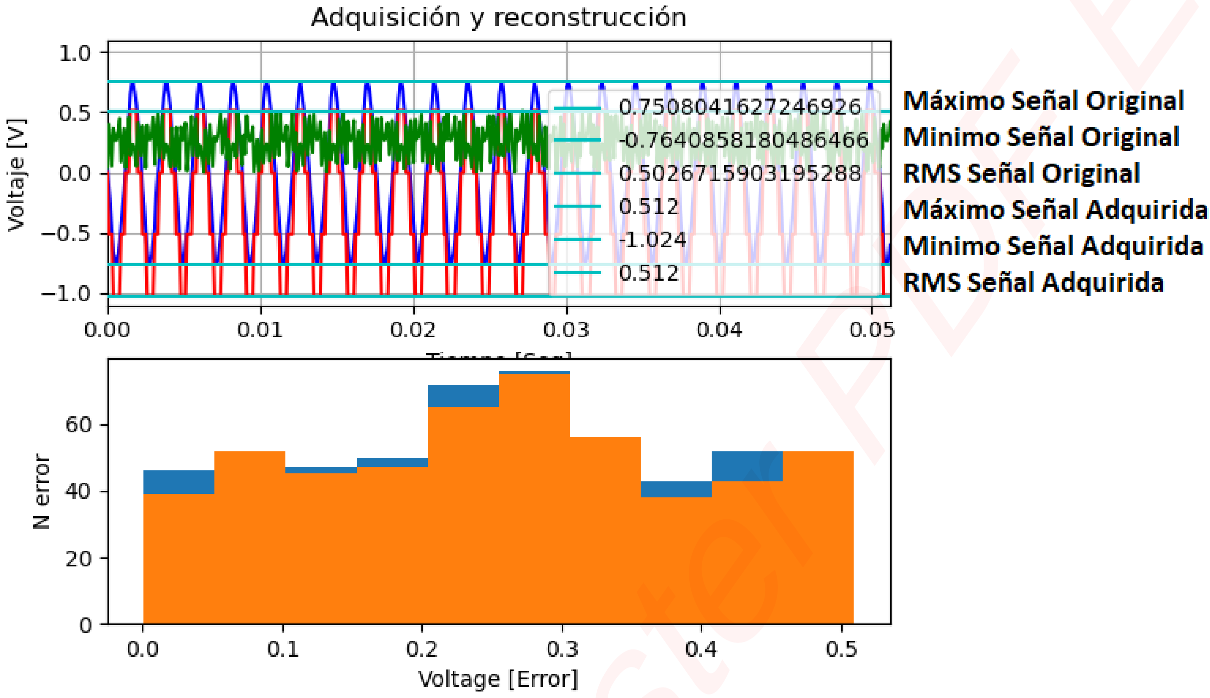
Señal Triangular con digitalización con 8 bits



Señal Triangular con digitalización con 4 bits



Señal Triangular con digitalización con 2 bits



En la señal triangular se aprecia que en todas las digitalizaciones el error es casi homogéneo, como también el error aumenta a medida que el número de bits de cuantificación disminuye.

Se puede concluir que la cuantificación si afecta la relación señal a ruido.

Esta esta un pocom mejor aunque es bastente raro que el error solo sea positivo siendo que para las otras señales el error era positivo y negativo.
Mi apreciacion en este punto es que si bien veo que has trabajado bastante para lograrlo la visualizacion de tus resultados no es facil de apreciar, este todo muy apretado y con trazos muy gruesos que dificultan ver. Pero mas importante, los histogramas de error no son consistentes en la mayoria de los casos y hay mucha variacion de escalas entre la señal generada y la capturada, casi del 50%, lo cual tambien es raro.
Creo que este punto merece que lo desarrolles mejor, con gragicos mas grandes y normalices el error en terminos de LSB y no def volts para que se mantenga consistente entre tus experimentos

Sistema de números

1. Diferencias entre la representación flotante de simple precisión de (32b) y el sistema de punto fijo Qn.m.

	Flotante Simple	**Punto fijo Qn.m**
Precisión	Mayor	Menor
Espacio entre números (GAP)	variable	constante
Determinismo	Menor	Mayor
Rango Dinámico	Mayor	Menor
Procesamiento	Mayor	Menor
Costo	Mayor	Menor

2. Se escribe los bits de los siguientes números decimales (o el más cercano) en float, Q1.15, Q2.14

2.1. Decimal **0.5**

- Signo *0b0*

  Exponente *0b01111110*

  Mantisa *1,= 00000000000000000000000*

  Representación en Hexadecimal de Flotante *0x3F000000*
- Representación en Hexadecimal Q1.15 *0x4000*
- Representación en Hexadecimal Q2.14 *0x2000*

2.2. Decimal **-0.5**

- Signo *0b1*

  Exponente *0b01111110*

  Mantisa *1,= 00000000000000000000000*

  Representación en Hexadecimal Flotante *0xBF000000*
- Representación en Hexadecimal Q1.15 *0xC000*
- Representación en Hexadecimal Q2.14 *0xA000*

2.3. Decimal **-1.25**

- Signo *0b1*

  Exponente *0b01111111*

  Mantisa *1,= 01000000000000000000000*

  Representación en Hexadecimal Flotante *0xBFA00000*
- Representación en Hexadecimal Q1.15 *No se puede representar por ser menor a -1*
- Representación en Hexadecimal Q2.14 *0xD000*

0xB000

2.4. Decimal **0.001**

- Signo *0b0*

  Exponente *0b 01110101*

  Mantisa *1,= 00000110001001001101110*

  Representación en Hexadecimal Flotante *0x3A83126E*
- Representación en Hexadecimal Q1.15 *0x0021*
- Representación en Hexadecimal Q2.14 *0x0010*

2.5. Decimal **-2.001**

- Signo *0b1*

  Exponente *0b 10001001*

  Mantisa *1,= 11110100010000000000000*

  0xC0001062

  Representación en Hexadecimal Flotante *0xC4FA2000*
- Representación en Hexadecimal Q1.15 *No se puede representar por ser menor a -1*
- Representación en Hexadecimal Q2.14 *El valor más cercano seria -2 0xC000*

0x8000

2.6. Decimal **204000000**

- Signo *0b0*

  Exponente *0b 10011010*

  Mantisa *1,= 10000101000110010110000*

  Representación en Hexadecimal Flotante *0x4D428CB0*
- Representación en Hexadecimal Q1.15 *No se puede representar por ser mayor a 1*
- Representación en Hexadecimal Q2.14 *No se puede representar por ser mayor a 2*

Apéndice.

Código en Python de la Adquisición y reconstrucción

```
In [ ]:
```

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

length = 512
fs = 10000

factorOrg = 1.024 / 32767
factorAdc = 1.024 / 512

header = b'header'
fig = plt.figure (1 )


adcAxe = fig.add_subplot (2,1,1)
time = np.linspace(0,length/fs,length)
plt.title("Adquisición y reconstrucción")
plt.xlabel("Tiempo [Seg]")
plt.ylabel("Voltaje [V]")
orgLn, adcLn, errLn, orgMaxLn, orgMinLn, orgRmsLn, adcMaxLn, adcMinLn, adcRmsLn = plt.p
lot ([],[],'b',[],[],'r',[],[],'g',[],[],'c',[],[],'c',[],[],'c',[],[],'c',[],[],'c',
[],[],'c')
adcAxe.grid ( True )
adcAxe.set_ylim (-1.1, 1.1)
adcAxe.set_xlim ( 0 ,length/fs )

hisAxe = fig.add_subplot (2,1,2)
plt.xlabel("Voltage [Error]")
plt.ylabel("N error")

def findHeader(f):
    index = 0
    sync = False
    while sync==False:
        data=b''
        while len(data) <1:
            data = f.read(1)
        if data[0]==header[index]:
            index+=1
            if index>=len(header):
                sync=True
        else:
            index=0

def readIntSignedFile(f):
    raw=b''
    while len(raw)<2:
        raw += f.read(1)
    return (int.from_bytes(raw[0:2],"little",signed=True))

def update(t):
    findHeader ( logFile )
    org = []
```

```
    adc = []
    err = []
    for chunk in range(length):
        orgN = readIntSignedFile(logFile) * factorOrg
        adcN = readIntSignedFile(logFile) * factorAdc
        errN = orgN - adcN
        org.append (orgN)
        adc.append (adcN)
        err.append (errN)

    orgLn.set_data ( time, org)
    adcLn.set_data ( time, adc )
    errLn.set_data ( time, err)

    ormMaxValue = readIntSignedFile(logFile) * factorOrg
    orgMaxLn.set_data ( time,np.full(length,ormMaxValue))
    orgMaxLn.set_label(ormMaxValue)

    orgMinValue = readIntSignedFile(logFile) * factorOrg
    orgMinLn.set_data ( time,np.full(length, orgMinValue))
    orgMinLn.set_label(orgMinValue)

    orgRmsValue = readIntSignedFile(logFile) * factorOrg
    orgRmsLn.set_data (time,np.full(length,orgRmsValue))
    orgRmsLn.set_label(orgRmsValue)

    adcMaxValue = readIntSignedFile(logFile) * factorAdc
    adcMaxLn.set_data ( time,np.full(length,adcMaxValue))
    adcMaxLn.set_label(adcMaxValue)

    adcMinValue = readIntSignedFile(logFile) * factorAdc
    adcMinLn.set_data ( time,np.full(length,adcMinValue))
    adcMinLn.set_label(adcMinValue)

    adcRmsValue = readIntSignedFile(logFile) * factorAdc
    adcRmsLn.set_data (time,np.full(length,adcRmsValue))
    adcRmsLn.set_label(adcRmsValue)

    legLg = adcAxe.legend(loc=4)
    plt.hist(err)

    return orgLn, adcLn, errLn, orgMaxLn, orgMinLn, orgRmsLn, adcMaxLn, adcMinLn, adcRm
sLn, legLg,


logFile=open("putty.log","w+b")
ani=FuncAnimation(fig,update,10,None,blit=True,interval=10,repeat=True)
plt.draw()
plt.show()
```

Codigo en C en la placa PsoC5 de Cypress CY8C5888LTI-LP097 con un Cortex-M3

main.c

```
/* ========================================
 *
 * Copyright JAIRO MENA, MAY 2020
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF jamenaso.
 *
 * ========================================
*/
#include "project.h"
#include "arm_math.h"
#include "math.h"
#include <stdbool.h>

#define DEVICE_0     0

#define NBYTES 2
#define HIGH 1
#define LOW 0
#define SAMPLES_LENGTH  512

#define LA_440 440
#define FS 10000
uint16_t f = LA_440;
uint16_t sweept = 1;
uint32_t tick = 0 ;
bool timer_flag;
int16_t sampleSig;
int8_t orgSig;
float t = 0;

float32_t ORG_float_V[SAMPLES_LENGTH] = {0};
float32_t ADC_float_V[SAMPLES_LENGTH] = {0};
int16_t sampleAdc(float* vector, uint16_t index);

float32_t maxValue,minValue, rms;
uint32_t maxIndex,minIndex = 0;

float32_t sinSig;
uint8_t square;

int16_t orgSaw;

double DoubleFrequAmpRetSawtooth(double x,double f,double amp);
void sendInt16(uint16_t value);

void USBUART_Configuration()
{
    USBUART_Start(DEVICE_0, USBUART_5V_OPERATION);
    while(!USBUART_bGetConfiguration());
```

Page 1 of 4

main.c

```
    USBUART_CDC_Init();
}

void ADC_Configuration()
{
    ADC_Start();
    ADC_StartConvert();
}

int main(void)
{
    uint32_t nSample;
    CyGlobalIntEnable;

    USBUART_Configuration();
    ADC_Configuration();
    Timer_ADC_Start();
    Timer_ADC_ISR_Start();
    //VDAC8_Start();
    WaveDAC8_1_Start();

    timer_flag = false;
    nSample = 0;

    while(true)
    {
        if(timer_flag)
        {
            timer_flag = false;

            //Señal Senoisoidal
            /*
            VDAC8_SetValue((uint8_t)((128*sin(t*f*2*PI))+128))
;
            ORG_float_V[nSample] = (float32_t)(128* sin(t*f*2*PI));
            sendInt16((int16_t)(128*sin(t*f*2*PI)));
            */

            //Señal Cuadrada
            /*
            t=((tick %(sweept*FS))/(float)FS);
            sinSig = sin(t*f*2*PI);
            if(sinSig > 0)
                square = 255;
            else
                square = 0;
            VDAC8_SetValue((uint8_t)square);
            ORG_float_V[nSample] = (float32_t)square - 127;
            sendInt16((int16_t)square - 127);
            */

            orgSaw = sampleAdc(ORG_float_V,nSample)
```

main.c

```
;
            sendInt16((int16_t)orgSaw);

            ADC_float_V[nSample] = (float32_t)(orgSaw>>14);
            sendInt16((int16_t)(orgSaw>>14));

            if(++nSample >= SAMPLES_LENGTH)
            {
                nSample = 0;

                arm_max_f32 ( ORG_float_V, SAMPLES_LENGTH, &maxValue,
&maxIndex );
                arm_min_f32 ( ORG_float_V, SAMPLES_LENGTH, &minValue,
&minIndex );
                arm_rms_f32 ( ORG_float_V, SAMPLES_LENGTH, &rms );

                sendInt16((int16_t)(maxValue));
                sendInt16((int16_t)(minValue));
                sendInt16((int16_t)(rms));

                arm_max_f32 ( ADC_float_V, SAMPLES_LENGTH, &maxValue,
&maxIndex );
                arm_min_f32 ( ADC_float_V, SAMPLES_LENGTH, &minValue,
&minIndex );
                arm_rms_f32 ( ADC_float_V, SAMPLES_LENGTH, &rms );

                sendInt16((int16_t)(maxValue));
                sendInt16((int16_t)(minValue));
                sendInt16((int16_t)(rms));

                USBUART_PutString("header");
                while(!USBUART_CDCIsReady()){}

                LED_Write(~LED_Read());
            }

            tick++;
        }
    }
}

int16_t sampleAdc(float* vector, uint16_t index)
{
    int16_t sample;
    while(!ADC_IsEndConversion(ADC_RETURN_STATUS)){};
    sample = (ADC_GetResult16()-32767);
    vector[index] = sample;
    return sample;
}

void sendInt16(uint16_t value)
{
```

main.c

```
    uint8_t d[NBYTES];
    d[HIGH] = (uint8_t)(value >> 8);
    d[LOW]  = (uint8_t)(value);
    USBUART_PutData(d,NBYTES);
    while(!USBUART_CDCIsReady()){}
}

void Timer_ADC_ISR_Interrupt_InterruptCallback()
{
    timer_flag = true;
    Timer_ADC_ReadStatusRegister();
    PIN_AUX_0_Write(~PIN_AUX_0_Read());
}
/* [] END OF FILE */
```

Configuración gráfica del PSoC