

Trabajo Practico N 1

10pts

In []:

Sistemas LTI. Demuestre si los siguientes sistemas son LTI:

1. $y(t) = x(t) * \cos(t)$

No es invariante en el tiempo, por lo tanto no es LTI. Contraejemplo:

Dado:

$$x(t) = t$$

$$x(t - t_0) = (t - t_0)$$

Entonces:



$$y_1(t - t_0) = (t - t_0) * \cos(t)$$

$$y_2(t - t_0) = (t - t_0) * \cos(t - t_0)$$

Por lo tanto:

$$y_1(t - t_0) \neq y_2(t - t_0)$$

2. $y(t) = \cos(x(t))$.

No es lineal, por lo tanto no es LTI. Contraejemplo:

Dado:

$$x(t) = t$$

Entonces:



$$y_1(t) = \cos(2 * t)$$

$$y_2(t) = 2 * \cos(t)$$

Para $t=0$:

$$y_1(0) = 1$$

$$y_2(0) = 2$$

Por lo tanto:

$$y_1(t) \neq y_2(t)$$

3. $y(t) = e^{x(t)}$

No es lineal por lo tanto no es LTI. Contraejemplo:

Dado:

$$x(t) = t$$

Entonces:

$$y_1(t) = e^{2*t}$$

$$y_2(t) = 2 * e^t$$

Para $t=0$:



$$y_1(0) = 1$$

$$y_2(0) = 2$$

Por lo tanto:

$$y_1(t) \neq y_2(t)$$

4. $y(t) = \frac{1}{2} * x(t)$

Si es LTI. Demostración:

Dado:

$$y_1(t - t_1) = \frac{1}{2} * x_1(t - t_1)$$

$$y_2(t - t_2) = \frac{1}{2} * x_2(t - t_2)$$



$$\frac{1}{2} * (a * x_1(t - t_1) + b * x_2(t - t_2)) = a * \left(\frac{1}{2} * x_1(t - t_1)\right) + b$$

$$* \left(\frac{1}{2} * x_2(t - t_2)\right)$$

$$a * \left(\frac{1}{2} * x_1(t - t_1)\right) + b * \left(\frac{1}{2} * x_2(t - t_2)\right) = a * y_1(t - t_1) + b * y_2(t - t_2)$$

Ruido de cuantización

1. Calcule la relación señal a ruido de cuantización teórica máxima de un sistema con un ADC de:

- 24 bits
- 16 bits
- 10 bits
- 8 bits
- 2 bits

Se calcula la S/N de cuantización máxima por ser un modelo específico ideal. Por ej. histograma de ruido uniforme (ruido blanco limitado en banda), comportamiento de señal sinusoidal pico a pico, etc. Ver cálculo en próxima celda.

2. Dado un sistema con un ADC de 10 bits, que técnica le permitiría aumentar la SNR? En que consiste? Si la señal está limitada en frecuencia y consideramos que la potencia de ruido se puede distribuir de manera uniforme en todo el espectro se puede considerar la densidad espectral de potencia de ruido como:



$$S_{\text{espectral}}(f) = \frac{P_q}{F_s}$$

Por lo tanto si la señal está limitada en frecuencia ($-F_s$ y $+F_s$) y tenemos la posibilidad de samplear a una mayor tasa, permitimos distribuir la potencia de ruido en un espectro mayor, mientras que sabemos que la señal tiene la característica definida. Esto se denomina Sobremuestreo. Por ej. si duplicamos la tasa de muestreo, dividimos en dos el espectro de ruido sobre el ancho de banda de la señal. Dado un sistema con un ADC de 10 bits y una SNR=62 dB, si duplicamos la tasa de muestreo disminuimos en 3 dB la potencia de ruido en la banda y aumentamos en 3 dB la relación S/N.

In [1]:

```
import numpy as np
import scipy.signal as sc

N = np.array([24, 16, 10, 8, 2])
SNRapprox = 1.76 + 6.02 * N
SNR = 10*np.log10(3/2)+10*np.log10(np.power(2,2*N))
for (i, r, r2) in zip(N, SNR, SNRapprox):
    print("SNR para N=%d\tSNR=%.2f db\tSNR (aprox) %.2f db" % (i, r, r2))
```

SNR para N=24	SNR=146.26 db	SNR (aprox) 146.24 db
SNR para N=16	SNR=98.09 db	SNR (aprox) 98.08 db
SNR para N=10	SNR=61.97 db	SNR (aprox) 61.96 db
SNR para N=8	SNR=49.93 db	SNR (aprox) 49.92 db
SNR para N=2	SNR=13.80 db	SNR (aprox) 13.80 db



Filtro antialias y reconstrucción

- Calcular el filtro antialias que utilizara para su practica y/o trabajo final y justifique su decision

El trabajo final consiste en la medición de parametros y características de ruido ambiental. Como ser maximo durante una ventana de tiempo, valor medio/rms, detección de desvios en frecuencia capturados desde la CIAA y visualizandolos en la PC. Por lo tanto la frecuencia de interes es por encima de los 100Hz (para evitar el ruido de 50Hz) hasta los 5khz. Considerando que la frecuencia máxima de sampleo transmitiendo por UART a 460800bps es del orden de los 11Khz, podemos utilizar un filtro antialias por hardware de primer orden respecto con frecuencia de corte 5khz. Calculando:

$$f_{corte} = 5kHz = \frac{1}{2 * PI * R * C}$$

$$R = 1Kohm$$

$$C = 33nF$$

Generación y simulación

1. Genere un modulo o paquete con al menos 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)

Se embebe la clase en el note a fines de poder ejecutar la misma, pero puede tranquilamente estar en un archivo a parte.

2. Realice los siguientes experimentos

- fs = 1000
 - N = 1000
 - fase = 0
 - amp = 1
- A. f0 = 0.1fs y 1.1fs Como podría diferenciar las senoidales?

Muestrear una señal de una frecuencia de 1.1khz a una tasa de muestreo de 1khz provoca perdida de información. Tenemos que aumentar la tasa demuestreo para poder diferenciarla.

A. f0 = 0.49fs y 0.51fs Como es la frecuencia y la fase entre ambas?

Al estar subsampleadas, se da el efecto que la fase son opuestas, pero en realidad tienen la misma fase y frecuencias relativamente parecidas. Si aumentamos la tasa de muestreo podemos ver mas detalle de la señal.

tip: Grafique los casos superponiendo la misma señal pero sampleada 10 veces mas

In [2]:

```

import numpy as np
import scipy.signal as sc

class mysignal():
    def sin(self, fs, f0, amp, samples, phase=0):
        n = np.arange(0, samples/fs, 1/fs)
        return amp*np.sin(2*np.pi*f0*n + phase)

    def square(self, fs, f0, amp, samples, phase=0):
        n = np.arange(0, samples/fs, 1/fs)
        return amp*sc.square(2*np.pi*f0*n + phase)

    def triang(self, fs, f0, amp, samples, phase=0):
        n = np.arange(0, samples/fs, 1/fs)
        return amp*sc.sawtooth(2*np.pi*f0*n + phase + np.pi/2, 0.5)

import matplotlib.pyplot as plt

N = 1000
fase = 0
amp = 1

# Ej 1.a.
# sample at 1khz
fs = 1000
fr = 0.1*fs # 100 Hz
fr2 = 1.1*fs # 1100Hz
t = np.arange(0, N/fs, 1/fs)
ms = mysignal()
r = ms.sin(fs, fr, amp, N)
r2 = ms.sin(fs, fr2, amp, N)

# plot
fig = plt.figure()
plt.title("Ej. 1. a. Tiempo de muestreo: "+str(fs))
tplot = np.arange(0, N/fs, 1/fs)
plt.plot(tplot, r, 'ro-')
plt.plot(tplot, r2, 'b^--')
# show first 25ms
plt.xlim(0, 0.025)
plt.xlabel("tiempo")

# sample at 10khz
fs= 10000
t = np.arange(0, N/(10*fs), 1/(10*fs))
r = ms.sin(fs, fr, amp, N)
r2 = ms.sin(fs, fr2, amp, N)

# plot
fig = plt.figure()
tplot = np.arange(0, N/fs, 1/fs)
plt.title("Ej. 1. a. Tiempo de muestreo: "+str(fs))
plt.plot(tplot, r, 'ro-')
plt.plot(tplot, r2)#, 'b^--')
# show first 25ms
plt.xlim(0, 0.025)
plt.xlabel("tiempo")

# Ej 1.b.

```

```

# sample at 1khz
fs = 1000
fr = 0.49*fs # Hz
fr2 = 0.51*fs # Hz
t = np.arange(0, N/fs, 1/fs)
r = ms.sin(fs, fr, amp, N)
r2 = ms.sin(fs, fr2, amp, N)

# plot
fig = plt.figure()
plt.title("Ej. 1. b. Tiempo de muestreo: "+str(fs))
tplot = np.arange(0, N/fs, 1/fs)
plt.plot(tplot, r, 'ro-')
plt.plot(tplot, r2, 'b^--')
# show first 25ms
plt.xlim(0, 0.025)
plt.xlabel("tiempo")

# sample at 10khz
fs= 10000
t = np.arange(0, N/(10*fs), 1/(10*fs))
r = ms.sin(fs, fr, amp, N)
r2 = ms.sin(fs, fr2, amp, N)

# plot
fig = plt.figure()
tplot = np.arange(0, N/fs, 1/fs)
plt.title("Ej. 1. b. Tiempo de muestreo: "+str(fs))
plt.plot(tplot, r, 'ro-')
plt.plot(tplot, r2)#, 'b^--')
# show first 25ms
plt.xlim(0, 0.025)
plt.xlabel("tiempo")

```

Out[2]:

Text(0.5, 0, 'tiempo')

Adquisición y reconstrucción con la CIAA

1. Genere con un tono de LA-440. Digitalice con 10, 8, 4 y 2 bits con el ADC, envíe los datos a la PC, grafique y comente los resultados
 - Señal original con su máximo, mínimo y RMS
 - Señal adquirida con su máximo, mínimo y RMS
 - Señal error = Original-Adquirida
 - Histograma del error
2. Realice el mismo experimento con una cuadrada y una triangular

Se sintetizó la señal sinusoidal desde la CIAA utilizando la DAC. Se realizó la adquisición simulando las distintas resoluciones haciendo uso del define BITS_ADC.

```

#include "sapi.h"

#define LENGTH 512

// scale 0-1023 to -512 to 511
#define ADC_OFFSET 512

// BITS ADC 10, 8, 4 o 2
#define BITS_ADC 10

// sample rate
#define FS 10000

int16_t adc [ LENGTH ];
int16_t signal;
uint16_t sample = 0;

uint32_t tick = 0;
uint16_t f = 440;
uint16_t B = 5000;
uint16_t swept = 1;
float t = 0;

int main ( void ) {
    boardConfig();
    uartInit(UART_USB, 460800);
    adcInit(ADC_ENABLE);
    dacConfig(DAC_ENABLE);

    cyclesCounterConfig( EDU_CIAA_NXP_CLOCK_SPEED );
    while(1) {
        cyclesCounterReset();

        uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[sample] ,sizeof(ad
c[0]));

        signal = ((int16_t )adcRead(CH1) - ADC_OFFSET );

        // capture with xx BITS_ADC
        signal = signal >> (10 - BITS_ADC) << (10 - BITS_ADC);
        adc[sample] = signal;

        t=((tick %(swept*FS))/(float)FS);

        dacWrite( DAC, 512*arm_sin_f32 (t*f*2*PI) + ADC_OFFSET);

        if ( ++sample==LENGTH ) { //22.7hz para 512
            sample = 0;
            uartWriteByteArray ( UART_USB ,"header" , 6 );
            gpioToggle(LED1);
        }
        tick++;
    }
}

```



```
        while(cyclesCounterRead()< 20400) //clk 204000000
            ;
    }
}
```

A continuación código de referencia en Python para visualizar lo solicitado. Se comentó la función de acuerdo al ejercicio y eventualmente se modificó el rango y de la señal error.

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import os
import scipy.signal as sc

N = 512
fs = 10000
f0 = 440
A = 512
header = b'header'

fig, [ origAxe, adcAxe, errorAxe, errorHistAxe ] = plt.subplots(4, 1)

adcAxe.grid ( True )
adcAxe.set_title("Señal adquirida")
adcAxe.set_ylim ( -520, 520 )
adcAxe.set_xlim ( 0 , N/fs )
adcAxe.set_xlabel("Tiempo")

origAxe.grid ( True )
origAxe.set_title("Señal original")
origAxe.set_ylim ( -520, 520 )
origAxe.set_xlim ( 0 , N/fs )
#origAxe.set_xlabel("Tiempo")

origAxe.grid ( True )
errorAxe.set_title("Error")
errorAxe.set_ylim (-520, 520)
errorAxe.set_xlim ( 0 , N/fs )
errorLine, = errorAxe.plot([],[])

time = np.linspace(0, N/fs, N)
adcLine, adcMaxLine, adcMinLine, adcRmsLine, = adcAxe.plot([],[], 'r', [],
[], 'b', [], [], 'b', [], [], 'g')

origLine, origMaxLine, origMinLine, origRmsLine, = origAxe.plot([], [],
[], [], 'b', [], [], 'b', [], [], 'g')
origSignal = A*np.sin(2*np.pi*f0*time)
#origSignal = A*sc.square(2*np.pi*f0*time)
#origSignal = A*sc.sawtooth(2*np.pi*f0*time + np.pi/2, 0.5)
origMaxValue = round(max(origSignal), 2)
origMinValue = round(min(origSignal), 2)
origLine.set_data (time, origSignal )
origMaxLine.set_data ( time, np.full(N, origMaxValue))
origMinLine.set_data ( time, np.full(N, origMinValue))
#origRmsValue = (origMaxValue - origMinValue)/2/2**(1/2)
origRmsValue = np.sqrt(np.mean(origSignal ** 2))
origRmsLine.set_data ( time, np.full(N, origRmsValue))

origMaxLine.set_label("Maximo "+str(origMaxValue))
origMinLine.set_label("Minimo " + str(origMinValue))

```

```
origRmsLine.set_label("RMS " + str(round(origRmsValue, 2)))
origLegend = origAxe.legend(loc='upper right')
```

```
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 readInt4File(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)
    adc = []
    for chunk in range(N):
        adc.append (readInt4File(logFile))

    negFound = False
    for i in range(N):
        if (adc[i] < 0):
            negFound = True
        if (negFound and adc[i] > 0):
            break

    adc[0:(N-i)] = adc[i:N]
    adc[N-i:N] = np.full(i, 0)

    adcMaxValue = max(adc)
    adcMinValue = min(adc)
    adcLine.set_data (time, adc)
    adcMaxLine.set_data(time, np.full(N, adcMaxValue))
    adcMinLine.set_data(time, np.full(N, adcMinValue))
    adcRmsValue = np.sqrt(np.mean(np.power(adc, 2)))
    #adcRmsValue = (adcMaxValue - adcMinValue)/2/2**(1/2)
    adcRmsLine.set_data (time, np.full(N, adcRmsValue))

    adcMaxLine.set_label("Maximo "+str(adcMaxValue))
    adcMinLine.set_label("Minimo " + str(adcMinValue))
    adcRmsLine.set_label("RMS " + str(round(adcRmsValue, 2)))
```

```

adcLegend = adcAxe.legend(loc='upper right')

errorSignal = origSignal-adc
errorLine.set_data(time, errorSignal)

errorHistAxe.clear()
errorHistAxe.set_title("Histograma error")
errorHistAxe.set_xlim(-512, +512)
errorHistAxe.hist(errorSignal)

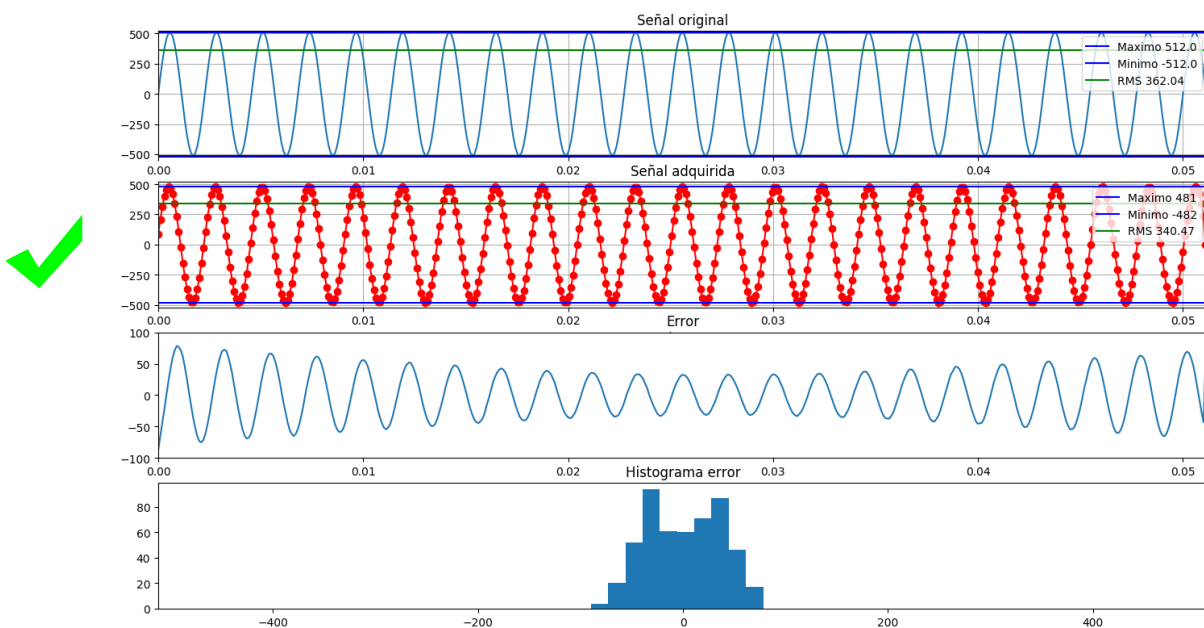
return adcLine, adcMaxLine, adcMinLine, adcRmsLine, adcLegend, errorLine,

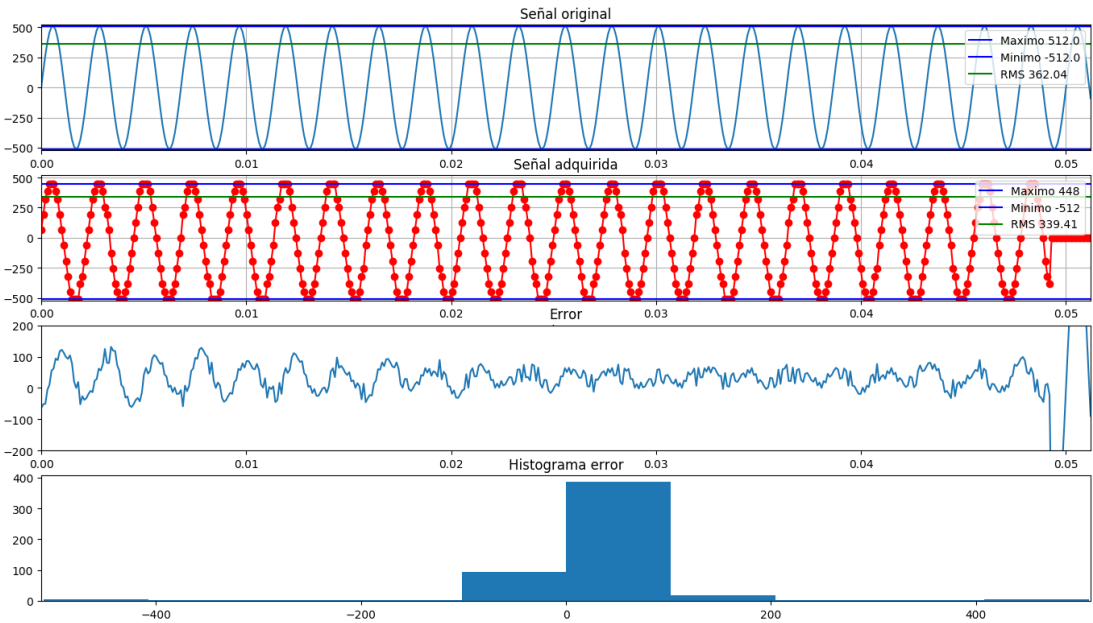
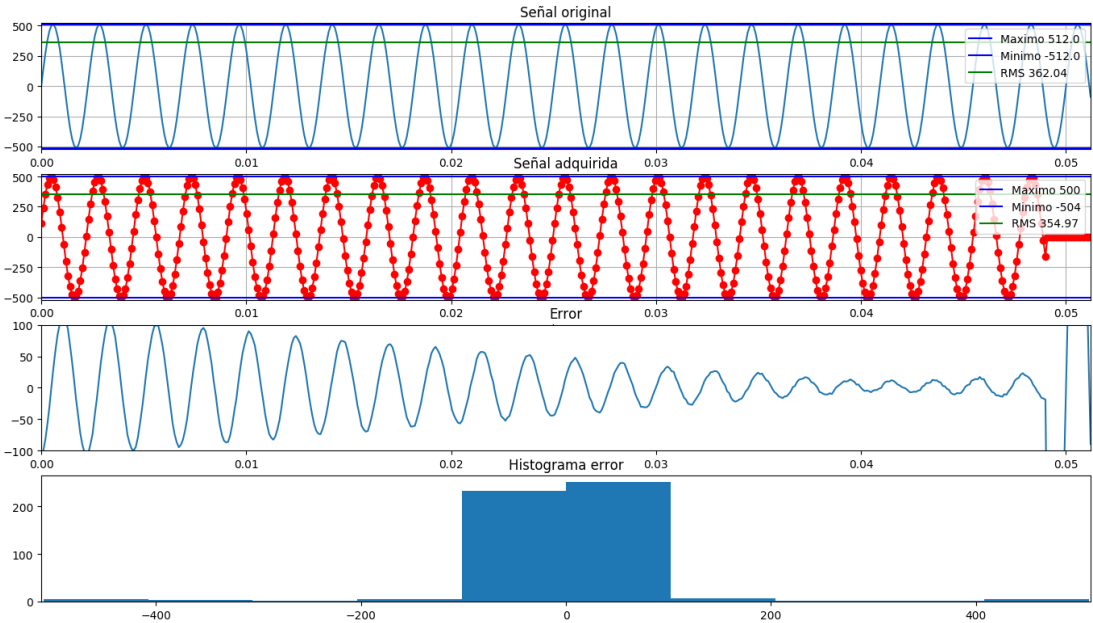
logFile = open("log.bin","w+b")
ani = FuncAnimation(fig, update, 1, None, blit=True, interval=2000, repeat=True)

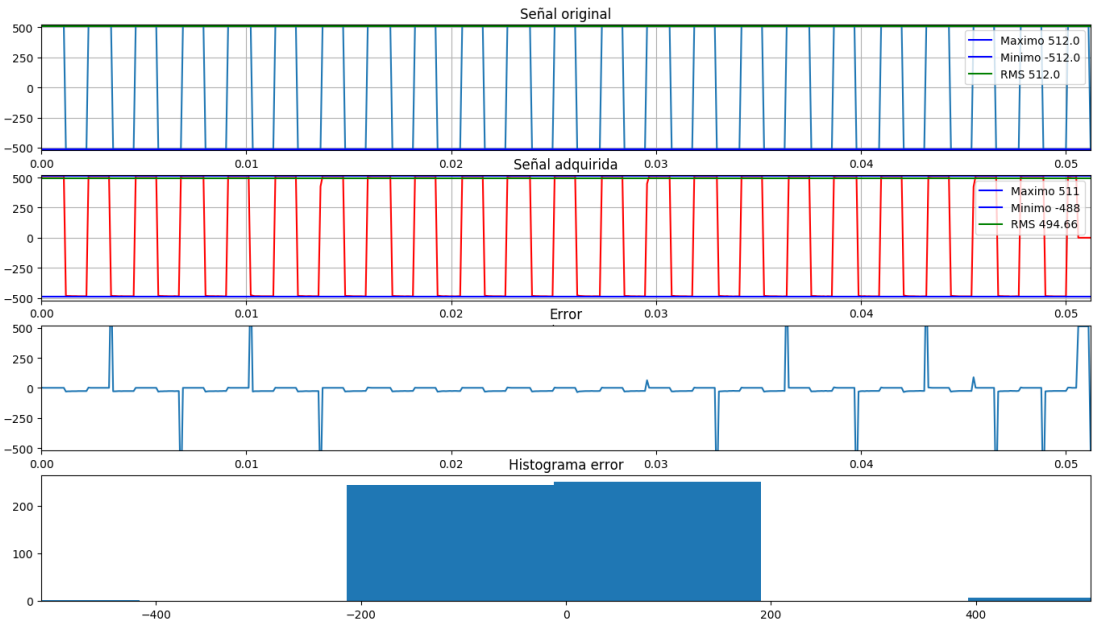
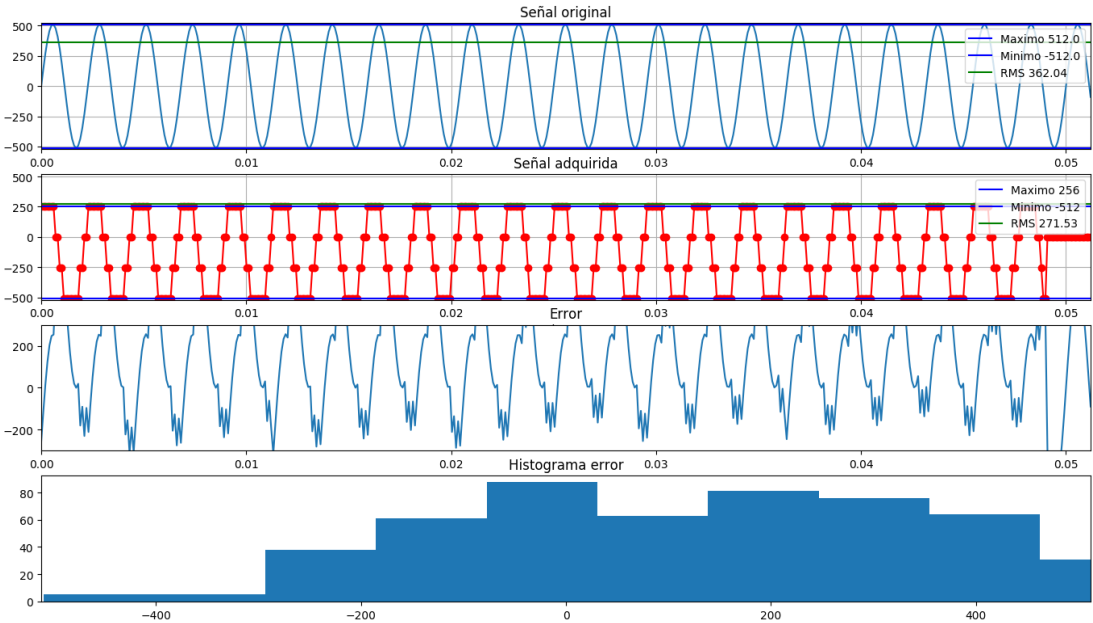
plt.draw()
plt.show()

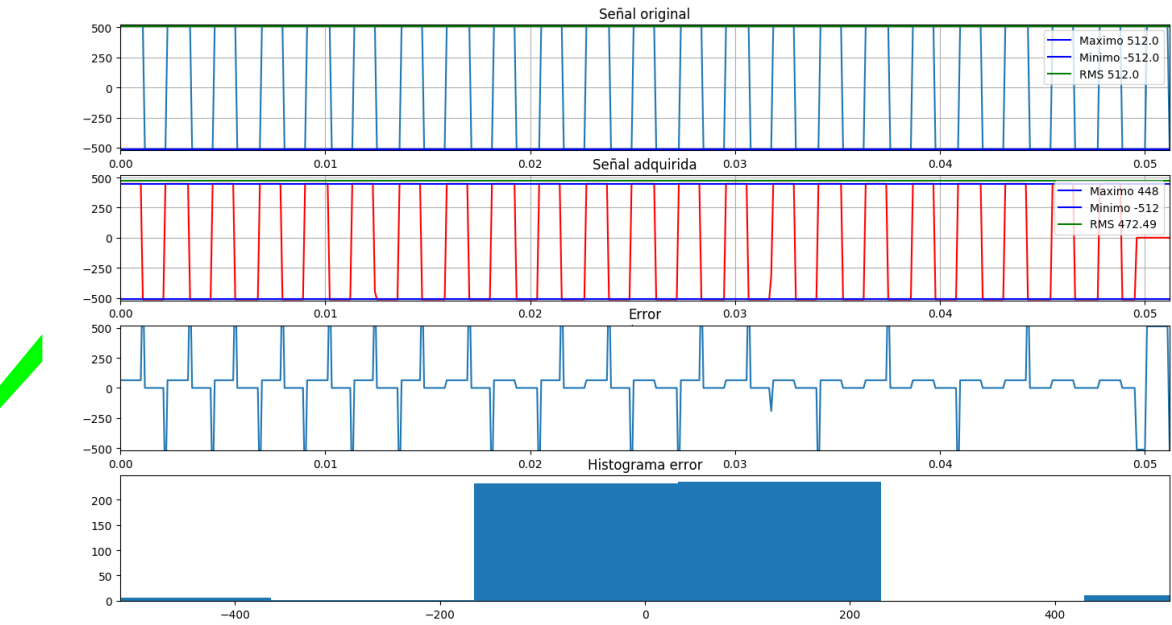
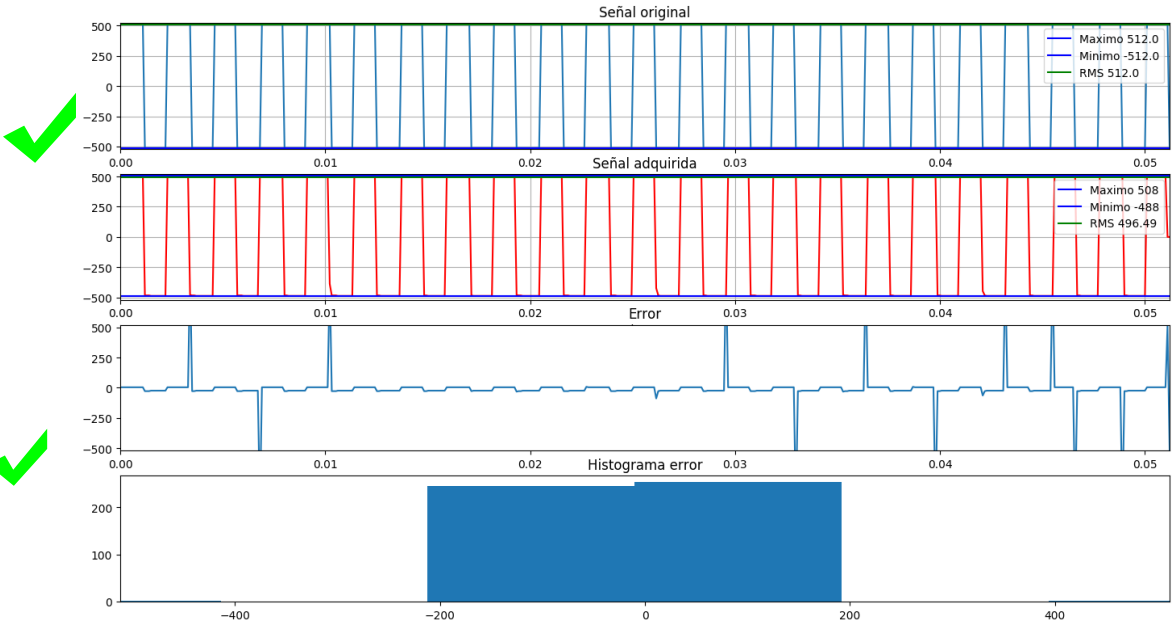
```

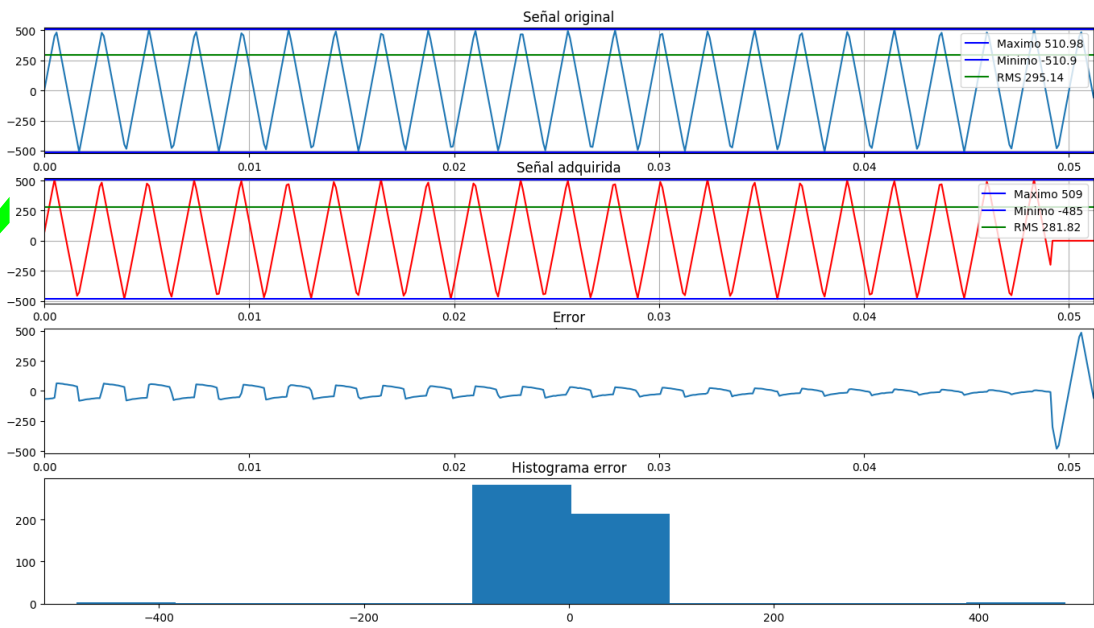
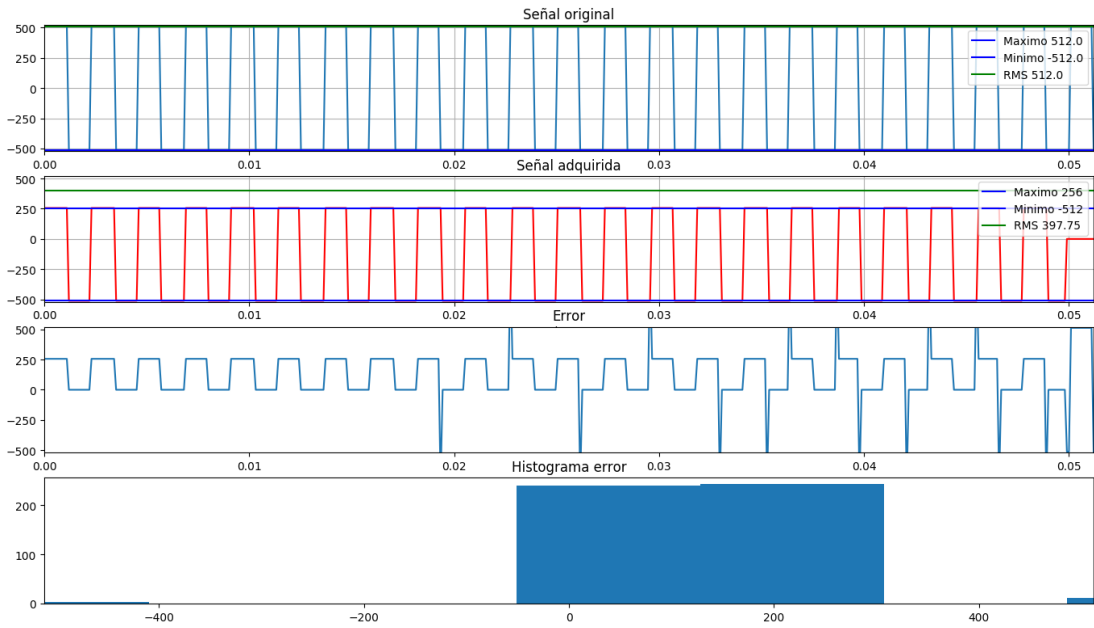
A continuación se muestra la salida de captura senoidal con una resolución de 10 Bits, 8 Bits, 4 Bits y 2 Bits. Se puede ver como aumenta el ruido de cuantización por la disminución de resolución. Adicionalmente vemos como baja el valor Rms de la señal.

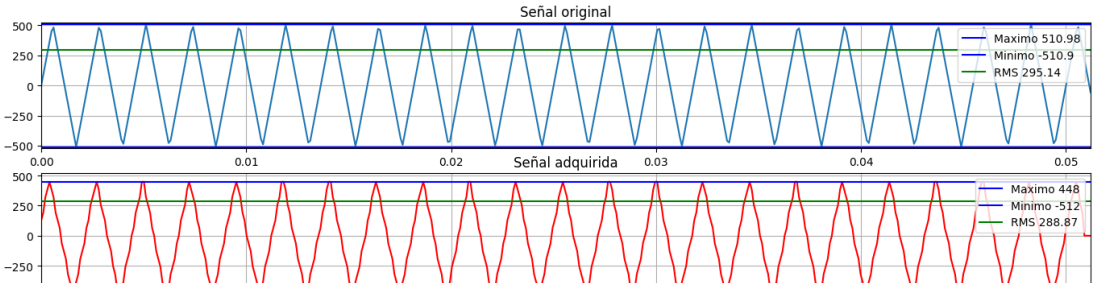
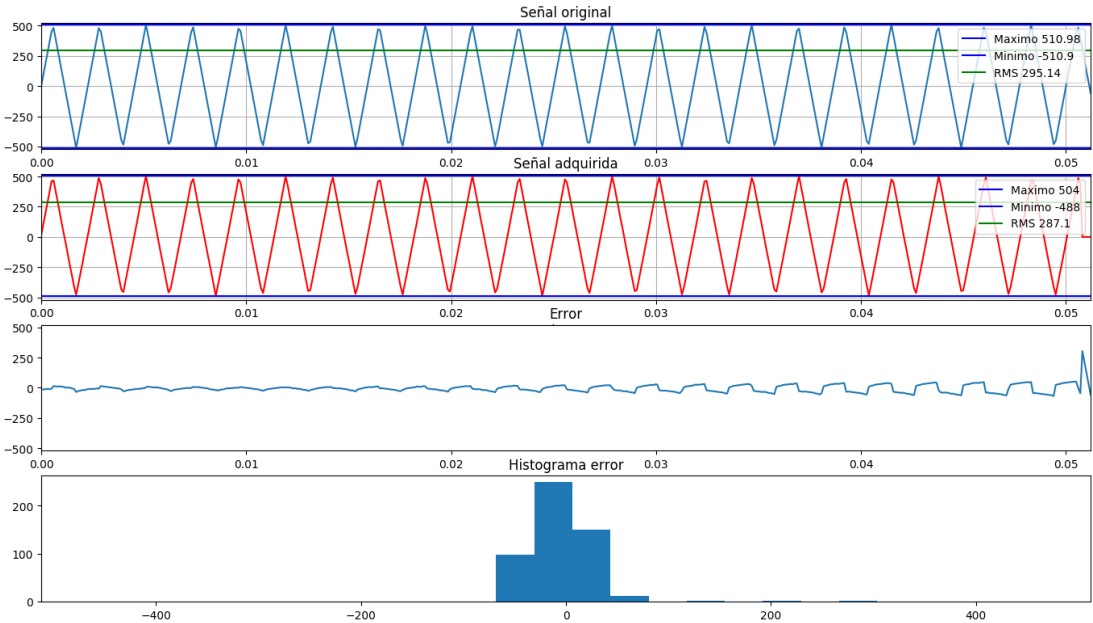












In []:

Sistema de números

1. Explique brevemente algunas de las diferencias entre la representación flotante de simple precision (32b) y el sistema de punto fijo Qn.m El uso del sistema Q permite trabajar con numeros decimales de forma un poco mas limitada pero no requiere el uso de unidad de punto flotante. Permite dividir la cantidad de bits en dos grupos, uno para la parte entera y otro para la parte decimal.
2. Escriba los bits de los siguientes números decimales (o el mas cercano) en float, Q1.15, Q2.14

- 0.5

float 0 0111 1110 000 0000 0000 0000 0000 0000

Comprobación: $-1^0 * 2^{126-127} * 1.0 = 0.5$

Q1.15 0.100 0000 0000 0000

Comprobación: $\frac{1}{2^1} = 0.5$

Q2.14 00.10 0000 0000 0000

Comprobación: $\frac{1}{2^1} = 0.5$

- -0.5

float 1 0111 1110 000 0000 0000 0000 0000 0000

Comprobación: $-1^1 * 2^{126-127} * 1.0 = -0.5$

Q1.15 1.100 0000 0000 0000

Comprobación: $-\frac{1}{2^1} = -0.5$

Q2.14 11.10 0000 0000 0000 Comprobación: $-\frac{1}{2} = -0.5$

- -1.25

float 1 0111 1111 010 0000 0000 0000 0000 0000

Comprobación: $-1^1 * 2^{127-127} * \left(1 + \left(\frac{1}{4}\right)\right) = -1.25$

Q1.15 1.000 0000 0000 0000 Comprobación: $-1 = -1$

Q2.14 10.11 0000 0000 0000

Comprobación: $-2 + \frac{1}{2} + \frac{1}{4} = -1.25$

- 0.001

float 0 0111 0101 000 0011 0001 0010 0110 1111

Comprobación: $-1^0 * 2^{117-127}$

$$* \left(1 + \left(\frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^{11}} + \frac{1}{2^{14}} + \frac{1}{2^{17}} + \frac{1}{2^{18}} + \frac{1}{2^{20}} + \frac{1}{2^{21}} + \frac{1}{2^{22}} + \frac{1}{2^{23}}\right)\right)$$

$$= .00100000004749745130$$

Q1.15 0.000 0000 0010 0001

Comprobación: $0 + \frac{1}{2^{10}} + \frac{1}{2^{15}} = 0.00100708007813$

Q2.14 00.00 0000 0001 0000

Comprobación: $0 + \frac{1}{2^{10}} = 0.0009765625$

- -2.001

float 1 1000 0000 000 0000 0001 0000 0110 0010

Comprobación: $-1^1 * 2^{128-127} * \left(1 + \left(\frac{1}{2^{11}} + \frac{1}{2^{17}} + \frac{1}{2^{18}} + \frac{1}{2^{22}}\right)\right) =$

$$-2.00099992752075195312$$

Q1.15 1.000 0000 0000 0000 = -1

Q2.14 11.00 0000 0000 0000 = -2

- 204000000

float 0 1001 1010 000 0010 0010 1100 1011 0000

Comprobación: $-1^1 * 2^{154-127}$

$$* \left(1 + \left(\frac{1}{2^1} \right) + \left(\frac{1}{2^6} \right) + \left(\frac{1}{2^8} \right) + \left(\frac{1}{2^{12}} \right) + \left(\frac{1}{2^{13}} \right) + \left(\frac{1}{2^{16}} \right) + \left(\frac{1}{2^{18}} \right) + \left(\frac{1}{2^{19}} \right) \right)$$

Faltarían los Q para este número grande

In [3]:

```
// float.c
// c code to check float representation
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    float f;
    unsigned int* ptr;
    int e, m, s;
    int i, first;
    char buf[512];
    buf[0] = NULL;

    if (argc < 2) {
        printf("Error\n");
        exit(-1);
    }

    sscanf(argv[1], "%f", &f);

    ptr = (unsigned int*)&f;

    // sign
    s = *ptr >> 31;
    // 8 bits exponent
    e = *ptr & 0x7f800000;
    e >= 23;
    // last 23 bits mantissa
    m = *ptr & 0x007fffff;

    printf("Signo: %x\n", s);
    printf("Exponente: Hexa: %x Decimal: %u\n", e, e);
    printf("Mantisa: %x\n", m);

    printf("Formula:\n");

    first = 1;
    for (i=1 ; i < 24; i++) {
        if (m & 1<=(23-i)) {
            if (!first)
                strcat(buf, " + ");
            else
                first = 0;
            sprintf(buf + strlen(buf), "1/2^%d", i);
        }
    }
    printf("(-1)^(%d) * e^(%d-127) * (1+ (%s) )\n", s, e, buf);
}
```

File "<ipython-input-3-e7e6ab0f5016>", line 1

// float.c

^

SyntaxError: invalid syntax

salida de comprobacion de float.c

```

$ ./float 0.5
Signo: 0
Exponente: Hexa: 7e Decimal: 126
Mantisa: 0
Formula:
 $(-1)^{(0)} * e^{(126-127)} * (1 + ( ) )$ 
$ ./float -0.5
Signo: 1
Exponente: Hexa: 7e Decimal: 126
Mantisa: 0
Formula:
 $(-1)^{(1)} * e^{(126-127)} * (1 + ( ) )$ 
$ ./float -1.25
Signo: 1
Exponente: Hexa: 7f Decimal: 127
Mantisa: 200000
Formula:
 $(-1)^{(1)} * e^{(127-127)} * (1 + (1/2^2) )$ 
$ ./float 0.001
Signo: 0
Exponente: Hexa: 75 Decimal: 117
Mantisa: 3126f
Formula:
 $(-1)^{(0)} * e^{(117-127)} * (1 + (1/2^6 + 1/2^7 + 1/2^{11} + 1/2^{14} + 1/2^{17} + 1/2^{18} + 1/2^{20} + 1/2^{21} + 1/2^{22} + 1/2^{23}) )$ 
$ ./float -2.001
Signo: 1
Exponente: Hexa: 80 Decimal: 128
Mantisa: 1062
Formula:
 $(-1)^{(1)} * e^{(128-127)} * (1 + (1/2^{11} + 1/2^{17} + 1/2^{18} + 1/2^{22}) )$ 
$ ./float 204000000
Signo: 0
Exponente: Hexa: 9a Decimal: 154
Mantisa: 428cb0
Formula:
 $(-1)^{(0)} * e^{(154-127)} * (1 + (1/2^1 + 1/2^6 + 1/2^8 + 1/2^{12} + 1/2^{13} + 1/2^{16} + 1/2^{18} + 1/2^{19}) )$ 

```

In []:


```

## python code to print float to Q signed numeric
def print_fixed(f, i, e):
    # check max
    if (f <= -2**(i-1)):
        # most negative number
        b = 2**(i+e-1)
    elif (f > (2**(i+e)-1) ):
        # most postive number
        b = 2**(i+e-1)-1
    else:
        # move decimal to integer limited to our decimal precision (e)
        a = f * (2**e)
        # round the decimal we cannot store
        b = int(round(a))
        # complement if negative
        if a < 0:
            # next three lines turns b into it's 2's complement.
            b = abs(b)
            b = ~b
            b = b + 1

    print("\nFloat: \t\t%f in Q%d.%d" % (f, i, e))
    print("Decimal: \t%d" % b)
    print("Hex: \t\t%x" % (b & (2**(e+i)-1)))
    print("Binary: \t%s" % format((b & (2**(e+i)-1)), '0'+str(e+i)+'b'))
    format(14, '#010b')

print_fixed(0.5, 1, 15)
print_fixed(0.5, 2, 14)
print_fixed(-0.5, 1, 15)
print_fixed(-0.5, 2, 14)
print_fixed(-1.25, 1, 15)
print_fixed(-1.25, 2, 14)
print_fixed(0.001, 1, 15)
print_fixed(0.001, 2, 14)
print_fixed(-2.001, 1, 15)
print_fixed(-2.001, 2, 14)
print_fixed(204000000, 1, 15)
print_fixed(204000000, 2, 14)

```



In []: