

June 19, 2020

1 Procesamiento de señales.

1.1 Trabajo Practico 1

1.1.1 Sistemas LTI

Demostrar que los sistemas son LTI

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

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

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

4. $y(t) = 1/2x(t)$

✓ 5. No es LTI, es lineal pero no invariante en el tiempo. $ay(t) = ax(t) * \cos(t) \mid \mid y(t-T_0) \neq x(t-T_0) * \cos(t-T_0)$

✓ 6. No es LTI, es invariante en el tiempo, pero no lineal. $ay(t) \neq \cos(ax(t)) \mid \mid y(t-T_0) = \cos(x(t-T_0))$

✓ 7. No es LTI, es invariante en el tiempo, pero no lineal. $ay(t) \neq e^{**} ax(t) \mid \mid y(t-T_0) = e^{**} x(t-T_0)$

✓ 8. Es LTI, es lineal e invariante en el tiempo. $ay(t) = 1/2ax(t) \mid \mid y(t-T_0) = 1/2x(t-T_0)$

Ejecutando el script de la siguiente celda, se ve en la primer columna un grafico que evalua un intervalo del sistema. En la segunda columna la entrada para cada sistema es multiplicada por dos, al tener autoescalado en los graficos, si el sistema fuese lineal, la imagen no deberia modificarse. En la tercer columna se ingresan los mismos valores pero en otro intervalo de 'tiempo' y si el sistema fuese invariable, la imagen deberia coincidir con la de la primer columna.

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

```
nroMuestras = 1000
t1 = np.arange(0,1000,1)
t2 = np.arange(500,1000,0.5)
```

```

X_entrada_sist_1= np.arange(0,1,1 / nroMuestras)
Y_salida_sist_1= X_entrada_sist_1 * np.cos(np.pi/180*t1/1)

X_entrada_sist_1L= np.arange(0,1,1 / nroMuestras) * 2
Y_salida_sist_1L= X_entrada_sist_1L * np.cos(np.pi/180*t1/1)

X_entrada_sist_1T= np.arange(0,1,1 / nroMuestras)
Y_salida_sist_1T= X_entrada_sist_1T * np.cos(np.pi/180*t2/1)


X_entrada_sist_2 = np.arange(0,1,1 / nroMuestras)
Y_salida_sist_2 = np.cos(2*np.pi*X_entrada_sist_2)

X_entrada_sist_2L = np.arange(0,1,1 / nroMuestras) * 2
Y_salida_sist_2L = np.cos(2*np.pi*X_entrada_sist_2L)


X_entrada_sist_3 = np.arange(0,2,2 / nroMuestras)
Y_salida_sist_3 = np.exp(X_entrada_sist_3)


X_entrada_sist_3L = np.arange(0,2,2 / nroMuestras) * 5
Y_salida_sist_3L = np.exp(X_entrada_sist_3L)


X_entrada_sist_4 = np.sin(2*np.pi*np.arange(0,1,1 / nroMuestras))
Y_salida_sist_4 = 1 / 2 * X_entrada_sist_4

X_entrada_sist_4L = (np.sin(2*np.pi*np.arange(0,1,1 / nroMuestras))) * 2
Y_salida_sist_4L = (1 / 2 * X_entrada_sist_4L)

X_entrada_sist_4L = (np.sin(2*np.pi*np.arange(0,1,1 / nroMuestras))) * 2
Y_salida_sist_4L = (1 / 2 * X_entrada_sist_4L)


fig = plt.figure()
graf1 = fig.add_subplot(4,3,1)
graf2 = fig.add_subplot(4,3,2)
graf3 = fig.add_subplot(4,3,3)

graf4 = fig.add_subplot(4,3,4)
graf5 = fig.add_subplot(4,3,5)
graf6 = fig.add_subplot(4,3,6)

graf7 = fig.add_subplot(4,3,7)
graf8 = fig.add_subplot(4,3,8)

```

```

graf9 = fig.add_subplot(4,3,9)

graf10 = fig.add_subplot(4,3,10)
graf11 = fig.add_subplot(4,3,11)
graf12 = fig.add_subplot(4,3,12)

graf1.plot(t1, X_entrada_sist_1)
graf1.plot(t1, Y_salida_sist_1)

graf2.plot(t1, X_entrada_sist_1L)
graf2.plot(t1, Y_salida_sist_1L)

graf3.plot(t2, X_entrada_sist_1T)
graf3.plot(t2, Y_salida_sist_1T)

graf4.plot(t1, X_entrada_sist_2)
graf4.plot(t1, Y_salida_sist_2)

graf5.plot(t1, X_entrada_sist_2L)
graf5.plot(t1, Y_salida_sist_2L)

graf6.plot(t2, X_entrada_sist_2)
graf6.plot(t2, Y_salida_sist_2)

graf7.plot(t1, X_entrada_sist_3)
graf7.plot(t1, Y_salida_sist_3)

graf8.plot(t1, X_entrada_sist_3L)
graf8.plot(t1, Y_salida_sist_3L)

graf9.plot(t2, X_entrada_sist_3)
graf9.plot(t2, Y_salida_sist_3)

graf10.plot(t1, X_entrada_sist_4)
graf10.plot(t1, Y_salida_sist_4)

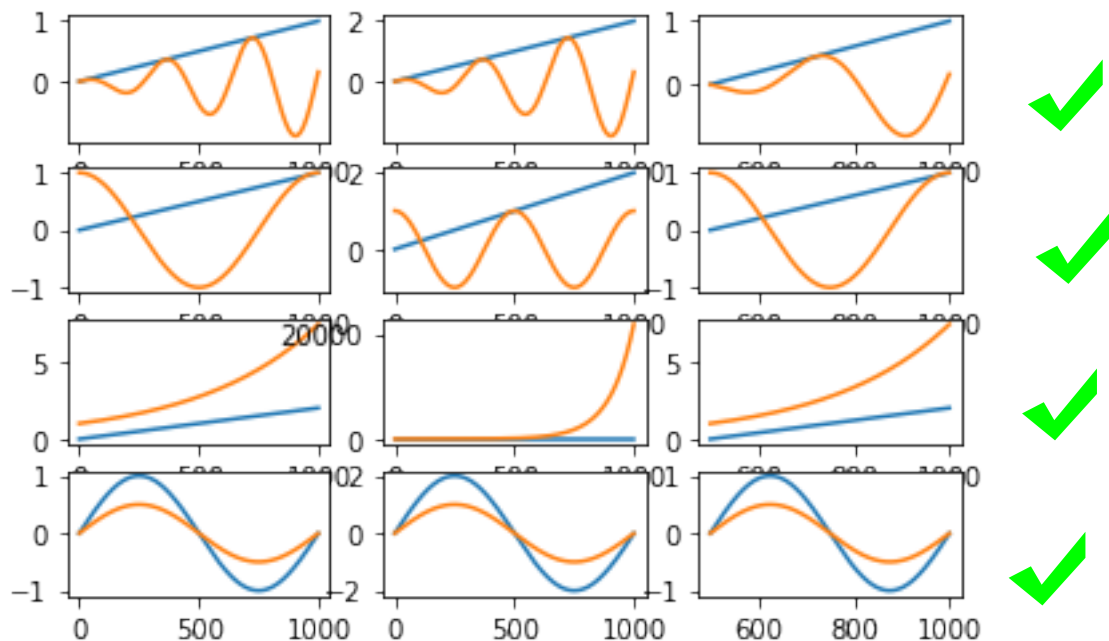
graf11.plot(t1, X_entrada_sist_4L)
graf11.plot(t1, Y_salida_sist_4L)

graf12.plot(t2, X_entrada_sist_4)
graf12.plot(t2, Y_salida_sist_4)

#signalC = np.sin(2*np.pi*signalFrec*tC)+0.5*np.sin(2*np.pi*210*tC)
#for i in range(len(fsD)):
#    contiAxe = fig.add_subplot(4,1,i+1)
#    plt.plot(tC,signalC,'r-',tC[:,fsC//fsD[i]],signalC[:,fsC//fsD[i]],'b-o')

```

```
#     contiAxe.set_ylabel(fsD[i])
plt.show()
```



Muy buena idea de dibujarlo, se ve perfecto, excelente!

1.1.2 Ruido de Cuantizacion

$$1. \text{ SNR} = 1,76 + 6,02 * N$$

Para 24 bits -> SNR = 146,24

Para 16 bits -> SNR = 98,08

Para 10 bits -> SNR = 61,96

Para 8 bits -> SNR = 49,92

Para 2 bits -> SNR = 13,8

2. Para aumentar la SNR se puede utilizar la tecnica de sobremuestreo (oversampling)


Por ejemplo, si sobremuestreo a 4X se obtiene 6dB extras. ✓

Observacion: Creo que en la diapositiva 41/42 del PDF correspondiente a la clase 2 hay un error, o tal vez yo lo estoy interpretando mal. Explica lo del sobre muestreo, es solo un detalle, dice que para un ADC de 10 bits tenemos 66dB de SNR y que se lleva a 72dB con el sobremuestreo 4X. Me parece que seria 61 a 67 dB, en vez de 66 a 71 dB.

1.1.3 Filtro antialias y reconstruccion

1. Calcular filtro.

Si, estaba mal, ya me lo habian comentado y lo subi al drive corregido, Pero muy bien que lo notastes.


$$f_c = 1 / (2 * \pi * R * C)$$

Para una frecuencia de sampleo de 10000 Hz, y una se#al de entrada de hasta 5000 Hz, se pretende un filtro con corte en 5000 Hz.

Para una resistencia de 1000 Ohms, el capacitor deberia ser de 31,85 nF.

Con un valor comercial de capacitor de 33 nF, la frecuencia de corte deberia rondar los 4825 Hz.

ACLARACION: Estoy suponiendo que el filtro que estamos instalando es eficaz, pero en la realidad un filtro de primer orden con solamente un capacitor y una resistencia, tiene una eficiencia baja, el recorte lo va logrando con una rampa muy estirada. ... Habria que poner un filtro de mejor calidad, o muestrear a una velocidad superior, ya que es mas 'economico' o 'simple' que un buen filtro, en este rango de frecuencias. ...



1.1.4 Generacion y simulacion

1. Modulo o paquete con funciones:

Se genero el paquete lz_proc_dig_sen que incluye las funciones solicitadas.

lz_proc_dig_sen **init.py** gen_sen.py

2. Se realizaron los experimentos.

2.1. En primer lugar, solo para darme cuenta lo que estaba pasando, y para distinguir las se#ales en el grafico, las defase algunos grados. Luego siguiendo el TIP del ejercicio y multiplicando por 10 la frecuencia de sampleo se ven las se#ales claramente.

2.2. En primer lugar la frecuencia de ambase se#ales se veia identica en el grafico, y su fase a 180 grados. Luego siguiendo el TIP del ejercicio y sampleando a 10 veces la velocidad inicial, se pudieron observar claramente las dos se#ales.

Aclaracion: por una cuestion de practicidad en esta JUPYTER NOTEBOOK no llamo al paquete, copie en la siguiente celda la funcion senoidal. Solo a fin de evitar que el profesor necesite instalar mi paquete. Se deben ejecutar las celdas en orden.

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

def senoidal(fs,f0,amp,muestras,fase):
    segundos = muestras / fs
    lineaTiempo = np.arange(0, segundos, segundos / muestras )
    signal = np.sin(2*np.pi * f0 * lineaTiempo + fase) * amp
    return signal

In [2]: #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon May 25 00:09:50 2020
@author: leo
"""

import matplotlib.pyplot as plt
```

```

import numpy as np

fig = plt.figure()

fig1 = fig.add_subplot(4,1,1)
fig2 = fig.add_subplot(4,1,2)
fig3 = fig.add_subplot(4,1,3)
fig4 = fig.add_subplot(4,1,4)

# CONSIGNA 2.1
signalC = senoidal(1000, 0.1 * 1000, 1, 1000, 0)
fig1.plot(signalC)

signalC = senoidal(1000, 1.1 * 1000, 1, 1000, np.pi / 18)
fig1.plot(signalC)

# TIP: SAMPLEAR 10X
signalC = senoidal(10000, 0.1 * 1000, 1, 1000, 0)
fig2.plot(signalC)

signalC = senoidal(10000, 1.1 * 1000, 1, 1000, 0)
fig2.plot(signalC)

# CONSIGNA 2.2
signalC = senoidal(1000, 0.49 * 1000, 1, 1000, 0)
fig3.plot(signalC)

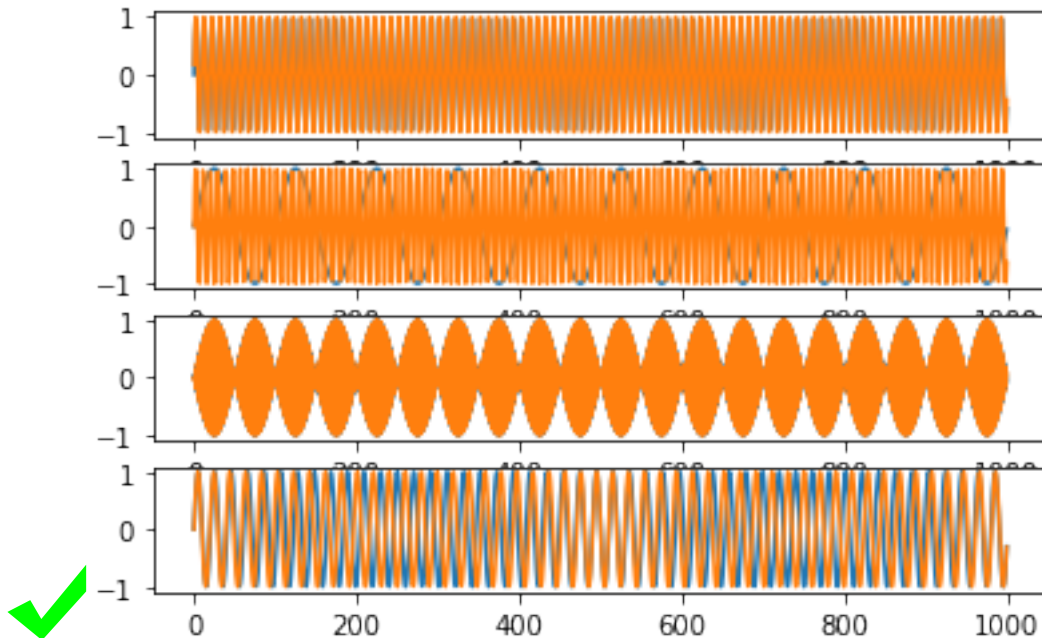
signalC = senoidal(1000, 0.51 * 1000, 1, 1000, 0)
fig3.plot(signalC)

# TIP: SAMPLEAR 10X
signalC = senoidal(10000, 0.49 * 1000, 1, 1000, 0)
fig4.plot(signalC)

signalC = senoidal(10000, 0.51 * 1000, 1, 1000, 0)
fig4.plot(signalC)

# plt.plot(tC, signalC, 'r-', tC[:, fs//fsD[i]], signalC[:, fs//fsD[i]], 'b-o')
# contiAxe.set_ylabel(fsD[i])
plt.show()

```



Aca podrias poner menos ciclos y que se apreciara mas la similitud o agregar un zoom

1.1.5 Adquisicion y reconstruccion con la CIAA

1. Genere Senoidal, Triangular (diente de sierra) y Cuadrada con el DAC de la CIAA
2. Digitalice con el ADC de la CIAA
3. Envie ambas se#ales, con sus maximos, minimos y rms por UART.
4. Arranque con 10 bits de resolucion y fui bajando, no incluyo todos los pasos porque recién a partir de 4 bits empieza a aparecer un error que se puede visualizar con la resolucion de pantalla que estoy usando y las imagenes que copio y pego.
5. Calculo la diferencia (error) con Python.

2 ACLARACION:

1. Calibre las resistencias de offset del ADC para tener la se#al lo mas centrada posible
2. Cuando configuro el ADC con resolucion aceptable, se ve poco el error y las se#ales directamente superpuestas.
3. El ejercicio pedia frecuencia de 440 Hz // A fin de que se vea fui bajando la frecuencia, sampleo en 200 Hz en 5000 samples.
4. Muestro solo un rango chiquito de tiempo, para que se aprecien un par de ciclos en la pantalla nomas.

5. MUESTRO LAS IMAGENES MAS SIGNIFICATIVAS, LO HICE A TODAS LAS RESOLUCIONES CON LAS TRES FORMAS DE ONDA

Tene en cuenta que en jupyter puedes incluir imagenes con un metalenguaje como html. puedes buscar en la web ejemplos. Solo para que lo tengas en cuenta

2.1 ADJUNTO DOC CON IMAGENES.

3 codigo CIAA

```
/=====
Autor: * Licencia: * Fecha: =====
// Inlcusiones
#include "sapi.h" #include "arm_math.h"
#define LENGTH 512 #define FS 10000 #define DELAY_FASE_US 0 #define
ADC_BITS_RESOLUCION 2
#define SENOIDAL // #define TRIANGULAR // #define CUADRADA
#define CICLOS_CLOCK_DELAY EDU_CIAA_NXP_CLOCK_SPEED / 1000000 * DE-
LAY_FASE_US #define CICLOS_CLOCK_ESPERA EDU_CIAA_NXP_CLOCK_SPEED / FS - CI-
CLOS_CLOCK_DELAY
int16_t adc[LENGTH]; int16_t dac[LENGTH]; uint16_t sample = 0; uint32_t tick = 0; uint16_t
f = 200; uint16_t B = 5000; uint16_t swept = 1; uint32_t maxIndex, minIndex = 0; q15_t max-
Value, minValue, rms = 0; uint32_t aux_signal = 0; float t = 0; int main ( void ) {

boardConfig ( );
uartConfig ( UART_USB, 460800 );
adcConfig ( ADC_ENABLE );
dacConfig ( DAC_ENABLE );

cyclesCounterInit ( EDU_CIAA_NXP_CLOCK_SPEED );

while(1) {
    cyclesCounterReset();

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

    //=====
    // GENERA SENOIDAL BASE DE TIEMPO
    t=((tick % (swept*FS))/(float)FS);

    //===== GENERA SENOIDAL =====
    #ifdef SENOIDAL
        dac[sample] = ((int16_t)(512 * arm_sin_f32 (t*f*2*PI) + 0)) << 0;
    #endif
    //=====

    //===== GENERA TRIANGULAR, DIENTE DE CIERRA =====
    #ifdef TRIANGULAR
        aux_signal = t * 1024 * f;
        dac[sample] = (aux_signal%1023) -512;
    #endif
    //=====
```



```

//===== GENERA CUADRADA =====
#ifdef CUADRADA
    aux_signal = ((int16_t)(512 * arm_sin_f32 (t*f*2*PI) + 0)) << 0 ;
    dac[sample] = (aux_signal >> 10) * 1023 + 512;
#endif
//=====

dacWrite( DAC, dac[sample] + 512); //tono
while(cyclesCounterRead() < CICLOS_CLOCK_ESPERA) //clk 204000000
    ;

adc[sample] = (((uint16_t)((adcRead(CH1)+( ((1<<(10-ADC_BITS_RESOLUCION))/2)    )) >> (10 -

if ( ++sample==LENGTH ) {

    // Calculos se#al
    arm_max_q15 ( adc, LENGTH, &maxValue,&maxIndex );
    arm_min_q15 ( adc, LENGTH, &minValue,&minIndex );
    arm_rms_q15 ( adc, LENGTH, &rms );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&maxValue ,2 );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&minValue ,2 );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&rms ,2 );

    // Calculos se#al
    arm_max_q15 ( dac, LENGTH, &maxValue,&maxIndex );
    arm_min_q15 ( dac, LENGTH, &minValue,&minIndex );
    arm_rms_q15 ( dac, LENGTH, &rms );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&maxValue ,2 );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&minValue ,2 );
    uartWriteByteArray ( UART_USB ,(uint8_t* )&rms ,2 );

    sample = 0;
    uartWriteByteArray ( UART_USB ,"header" ,6 );
    gpioToggle ( LEDG );
}
tick++;

}

}

In [2]: import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import os
##matplotlib qt

```

```

%matplotlib notebook
#%matplotlib inline
length = 512
fs = 5000
header = b'header'
fig = plt.figure (1 )

time = np.linspace(0,length/fs,length)

#adcLn, dacLn, maxLn, minLn, rmsLn, = plt.plot ([],[],'r',[],[],'y',[],[],'b',[],[],'b'

dacAxe = fig.add_subplot ( 4,1,1 )
dac_1, max_dac1, min_dac1, rms_dac1, = plt.plot ([],[],'r', [],[],'b',[],[],'b',[],[],

adcAxe = fig.add_subplot ( 4,1,2 )
adc_1, max_adc1, min_adc1, rms_adc1, = plt.plot ([],[],'r', [],[],'b',[],[],'b',[],[],

errorAxe = fig.add_subplot ( 4,1,3 )
dac_2, adc_2, errorLn, = plt.plot ([],[],'r', [],[],'b',[],[],'g')

adcAxe.grid ( True )
adcAxe.set_ylim ( -550 ,550 )
adcAxe.set_xlim ( 0 ,length/fs )

dacAxe.grid ( True )
dacAxe.set_ylim ( -550 ,550 )
dacAxe.set_xlim ( 0 ,length/fs )

errorAxe.grid ( True )
errorAxe.set_ylim ( -550 ,550 )
errorAxe.set_xlim ( 0 ,length/fs )

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 = []
    dac = []

    for chunk in range(length):
        adc.append (readInt4File(logFile))
        dac.append (readInt4File(logFile))

    adc_1.set_data ( time,adc )
    max_adc1.set_data ( time,np.full(length,readInt4File(logFile)))
    min_adc1.set_data ( time,np.full(length,readInt4File(logFile)))
    rms_adc1.set_data ( time,np.full(length,readInt4File(logFile)))
    # rms_adc1.set_label(rmsValue_adc)
    # rms_adc1 = adcAxe.legend()

    dac_1.set_data ( time,dac )
    max_dac1.set_data ( time,np.full(length,readInt4File(logFile)))
    min_dac1.set_data ( time,np.full(length,readInt4File(logFile)))
    rms_dac1.set_data ( time,np.full(length,readInt4File(logFile)))
    # rms_dac1.set_label(rmsValue_dac)
    # rms_dac1 = dacAxe.legend()

    dac_2.set_data ( time,dac )
    adc_2.set_data ( time,adc )
    errorLn.set_data ( time, list(np.array(dac)-np.array(adc)) )

    histo = fig.add_subplot ( 4,1,4 )
    plt.hist(np.array(dac)-np.array(adc))

    return dac_1, max_dac1, min_dac1,rms_dac1,adc_1,max_adc1,min_adc1,rms_adc1,dac_2,a

```

```

logFile=open("/home/leo/Documentos/log.bin","w+b")
update(0)
## SI ACTIVO LA LINEA DE CODIGO SIGUIENTE, MUESTRA LAS SE#ALES Y ERROR ONLINE
## EN FORMA ANIMADA, INCLUSO EL HISTOGRAMA, PERO EL HISTOGRAMA LO VA PISANDO.
#ani=FuncAnimation(fig,update,1,None,blit=False,interval=10,repeat=False)
plt.draw()
plt.show()

```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

3.1 Sistema de numeros

1. Diferencias entre flotante simple y Qn.m

La representacion en punto flotante permite GAP variable entre numeros, aumentando su rango dinamico. Llega a tener un GAP 10 millones de veces mas chico que el numero. En punto fijo esto no sucede, pero el costo de su utilizacion es menor, en consumo de CPU y todas las operaciones.

El punto flotante requiere procesadores mas potentes.

2. Conversion de numeros.

0.5 Float: 00111111000000000000000000000000 Q1.15: 0100000000000000 Q2.14: 0010000000000000 ✓

-0.5: Float: 10111111000000000000000000000000 Q1.15: 1100000000000000 Q2.14: 1010000000000000

1010000000000000

da 0xE0 00 vos pusistes -1.5

-1.25 Float: 10111111010000000000000000000000 Q1.15: Fuera de rango: Se puede representar de -1 a + 0,99999... Q2.14: 1000000000000000 **da 0xB0 00 vos pones -2**

0.001 Float: 00111010100000110001001001101111 Q1.15: 0000000000100000 Q2.14: 0000000000010000 ✓

-2.001 Float: 11000000000000000001000001100010 Q1.15: Fuera de rango: Se puede representar de -1 a + 0,99999... Q2.14: Fuera de rango: Se puede representar hasta -2 ✓

204000000 Float: 01001101010000101000110010110000 Q1.15: No se puede representar Q2.14: No se puede representar ✓