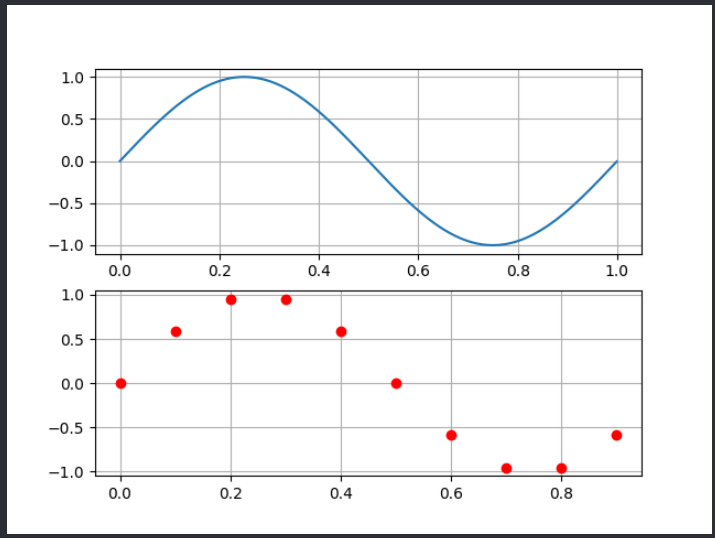
FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Procesamiento de señales, fundamentos

Maestría en sistemas embebidos
Universidad de Buenos Aires
MSE 5Co2020

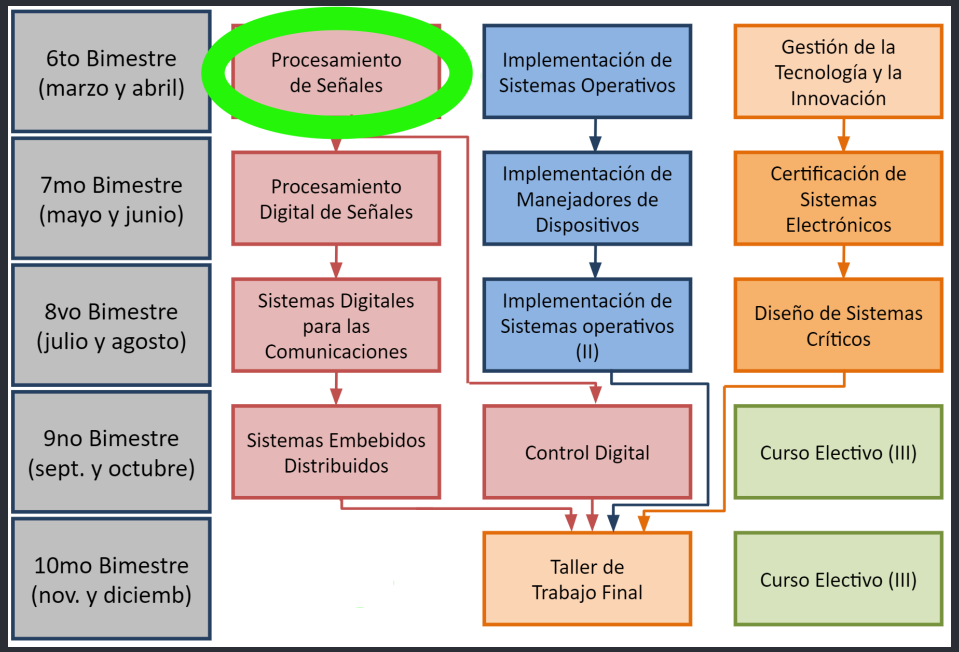
Clase 1 - Introducción



Ing. Pablo Slavkin
slavkin.pablo@gmail.com
wapp:011-62433453

Plan de vuelo

Ud. Está aquí

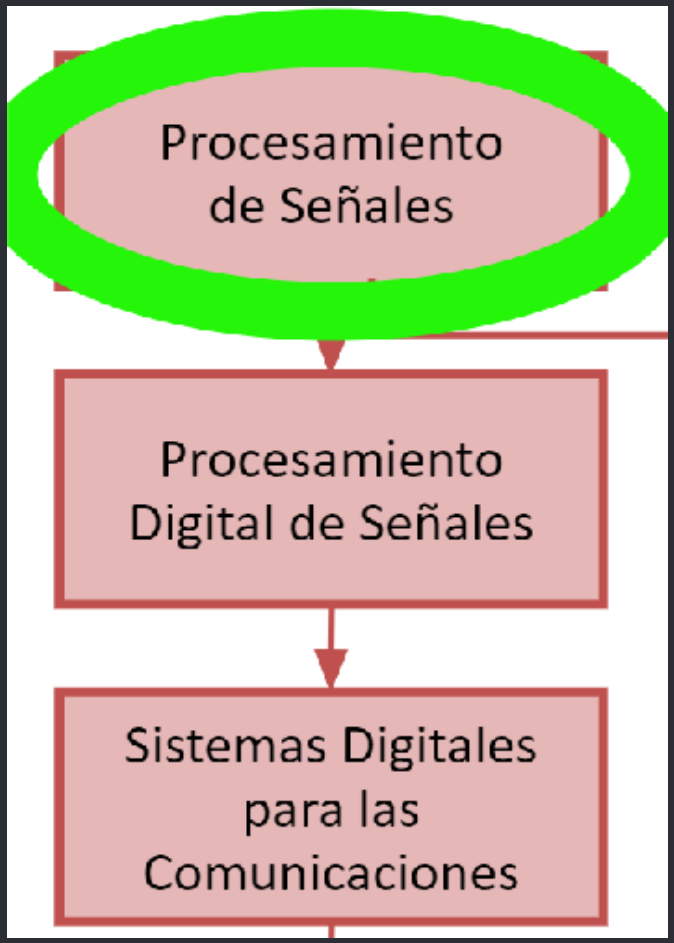


Colaboradores

- Gonzalo Lavigna <gonzalolavigna@gmail.com>
- Guillermo Guichal <guillermo.guichal@gmail.com>
- Federico Giordano Zacchigna <federico.zacchigna@gmail.com>

Plan de vuelo

Ud. Esta aquí



1.
   - Python, numpy
   - CIAA
   - Sampleo
   - Fourier, DFT
2.
   - Python, numpy
   - CIAA
   - VHDL, FPGA
   - Filtrado y ventaneo
3.
   - Python
   - VHDL, FPGA
   - Comunicaciones
   - Implementación
   - Hi-Speed

Bibliografía

Libros, links y otro material

[1] Steven W. Smith.
The Scientist and Engineer's Guide to Digital Signal Processing
Second Edition, 1999.

[2] Allen B. Downey
Think DSP - Digital Signal Processing in Python

[3] Richard Lyons.
Understanding digital signal processing.
Third edition.

[4] Boaz Porat.
Digital Processing of Random Signals: Theory and Methods. *Digital Processing of Random Signals: Theory and Methods*.

[5] Allen B. Downey
Think Python, 2nd Edition, - How to Think Like a Computer Scientist

[6] Emmanuel C Ifeachor, Barrie W Jervis
Digital Signal Processing. *A practial approach*.

[7] NW. Taylor, Francis Group, LLC.
Introduction to Python Programming.

[8] Matt Harrison
Illustrated guide to python 3

Enuestas

Encuesta anónima clase a clase

Propiciamos este espacio para compartir sus sugerencias, criticas constructivas, oportunidades de mejora y cualquier tipo de comentario relacionado a la clase.

Encuesta anónima

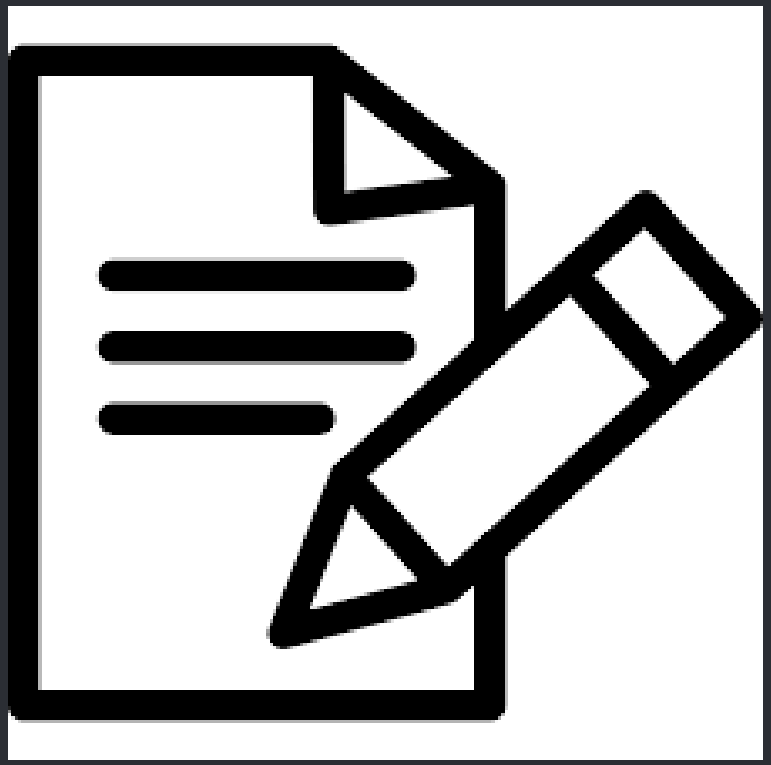
https://forms.gle/1j5dDTQ7qjVfRwYo8

Link al material de la material

https://drive.google.com/drive/u/1/folders/1TlR2cgDPchL_4v7DxdpS7pZHtjKq38CK

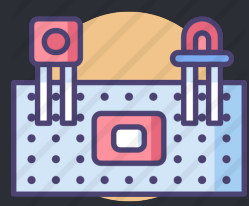
Metodo de evaluacion

- 3 pts - Examen
- 3 pts - TP Python
- 4 pts - Proyecto final



Evaluación

Proyecto final



- Deberá incluir algún tipo de procesamiento en hardware. ej. DFT, FIR, IIF, etc.
- Puede utilizar el ADC para samplear, DAC para reconstruir y/o canales de comunicación para adquirir datos previamente digitalizados
- Presentación de 10 minutos.
- Deberá funcionar!

Ejemplos:

- Filtrado y/o procesamiento de audio, señales biomédicas, etc.
- Técnicas de compresión en dominio de la frecuencia
- Aplicaciones con acelerómetro, magnetómetro, T+H

porque digital?

digital vs analógico

- digital
  - Reproducibilidad
  - Tolerancia de componentes
  - Partidas todas iguales
  - Componentes no envejecen
  - Fácil de actualizar
  - Soluciones de un solo chip
- analógico
  - Gran rango dinamico de entrada
  - Alto ancho de banda
  - Alta potencia
  - Baja latencia

Señales y sistemas

Que son?

Señal

Una señal, en función de una o más variables, puede definirse como un cambio observable en una entidad cuantificable

Sistema

Un sistema es cualquier conjunto físico de componentes que actúan en una señal, tomando una o más señales de entrada, y produciendo una o más señales de salida.

Señales y sistemas

Tipos de señales

- De tiempo continuo
- Pares
- Periódicas
- De energía
- Reales

- De tiempo discreto
- Impares
- Aperiódicas
- De potencia
- Imaginarias

Señales y sistemas

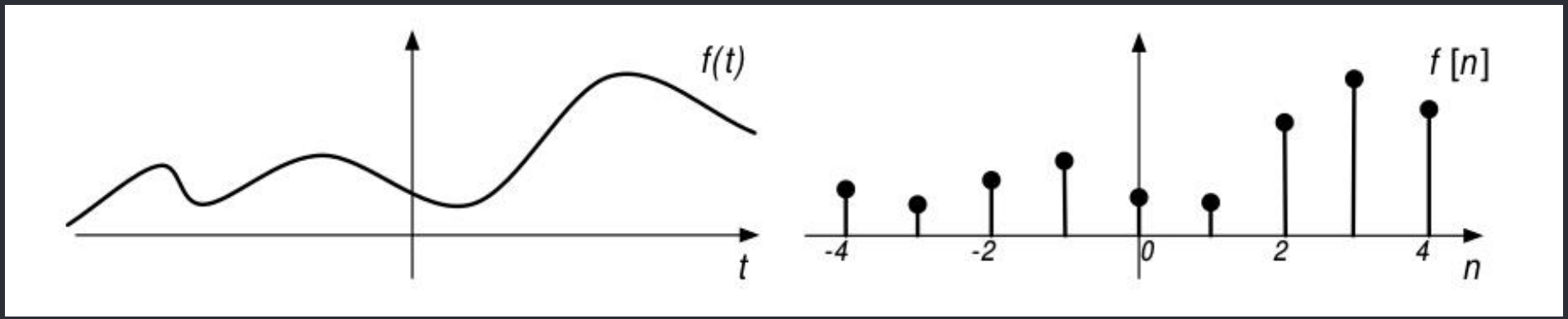
Tipos de señales

- De tiempo continuo

Tiene valores para todos los puntos en el tiempo en algún intervalo (posiblemente infinito)

- De tiempo discreto

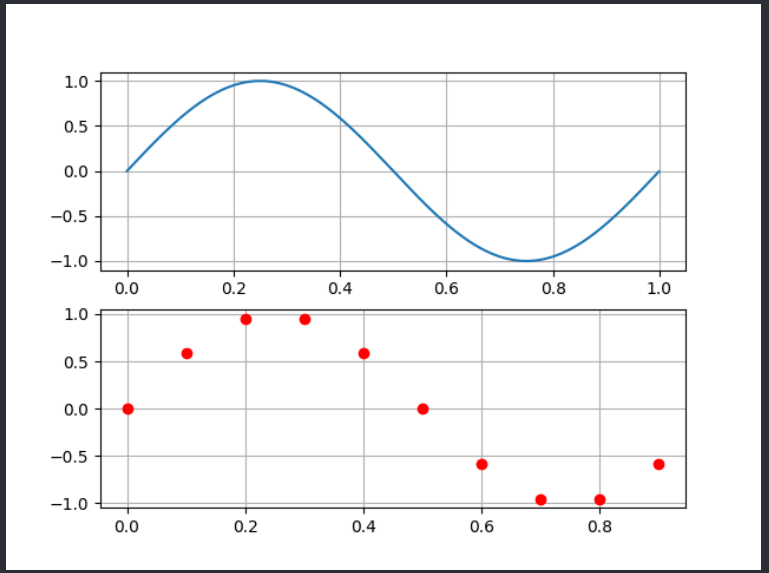
Tiene valores solo para puntos discretos en el tiempo



Generación de señales en Python

Continuo? vs discreto

```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(1)
Nc=1000
tc = np.linspace(0, 1, Nc)
ax1 = fig.add_subplot(2,1,1)
ax1.plot(tc, np.sin(2*np.pi*tc),"b-")
ax1.grid(True)

Nd=10
td = np.linspace(0, 1, Nd)
ax2 = fig.add_subplot(2,1,2)
ax2.plot(td, np.sin(2*np.pi*td),"ro")
ax2.grid(True)

plt.show()
```



Podrían pensarse como muestras de una señal de tiempo continuo

$x[n] = x(nT)$ donde n es un número entero y **T** es el período de muestreo.

Señales periódicas

Continua periódica

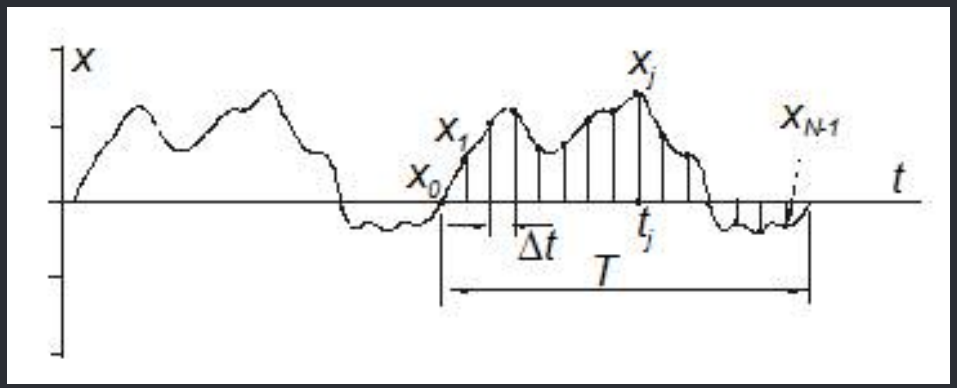
si existe un $T_0 > 0$, tal que $x(t + T_0) = x(t)$, para todo t

T_0 es el período de $x(t)$ medido en tiempo, y $f_0 = 1/T_0$ es la frecuencia fundamental de $x(t)$

Discreta periódica

si existe un entero $N_0 > 0$ tal que $x[n + N_0] = x[n]$ para todo n

N_0 es el período fundamental de $x[n]$ medido en espacio entre muestras y $F_0 = \Delta t/N_0$ es la frecuencia fundamental de $x[n]$



Sistemas

Sistema

Un sistema es cualquier conjunto físico de componentes que actúan en una señal, tomando una o más señales de entrada, y produciendo una o más señales de salida.

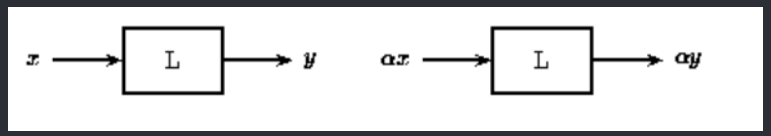
En ingeniería, a menudo la entrada y la salida son señales eléctricas.

Linealidad

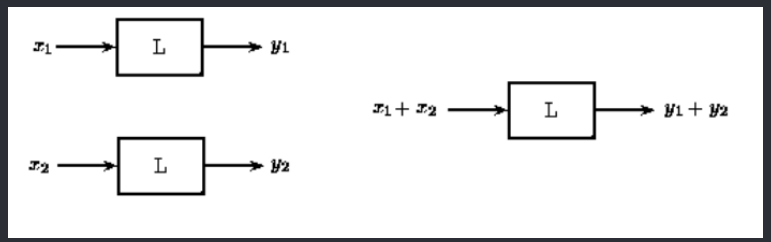
Un sistema es lineal cuando su salida depende linealmente de la entrada.

Satisface el principio de superposición.

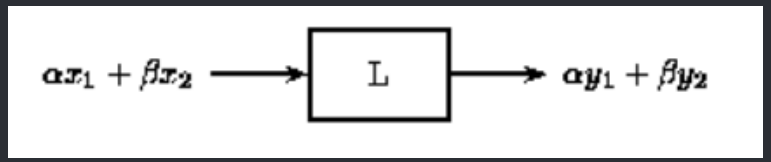
escalado



adición



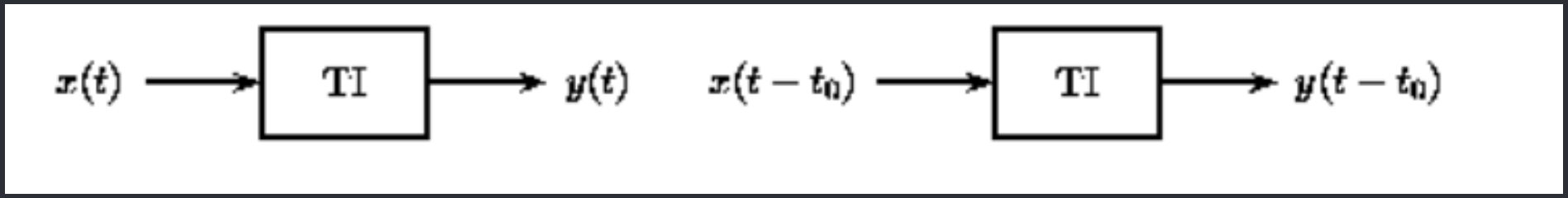
superposición



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

Invariantes en el tiempo

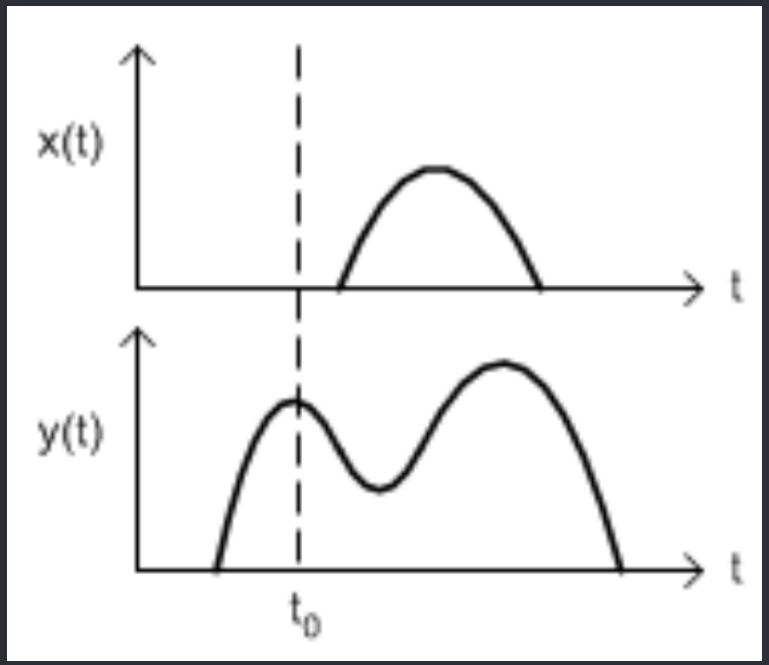
Un sistema es invariante en el tiempo cuando la salida para una determinada entrada es la misma sin importar el tiempo en el cual se aplica la entrada



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

Sistema causal

Un sistema es causal cuando la salida depende solo de los valores presentes y pasados de la entrada

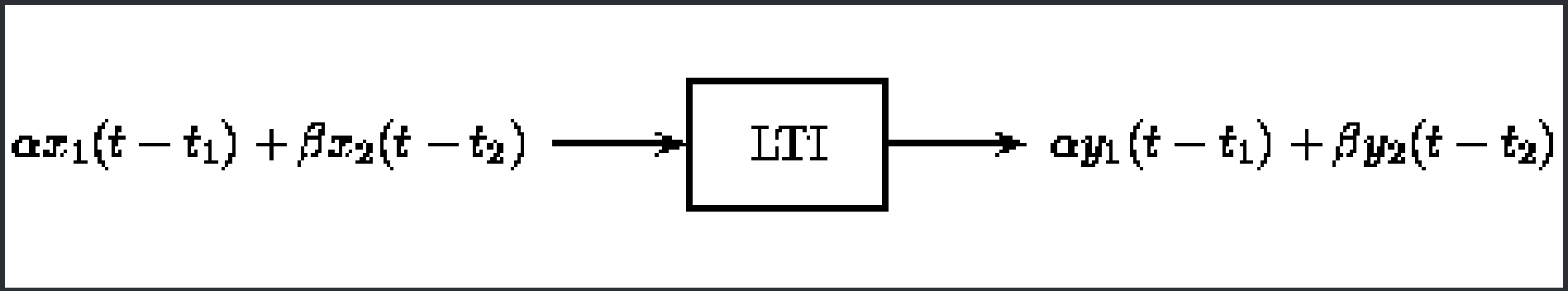


$y(t) = x(t+1)$ $y(t) = x(t-2)$

Sistemas

Lineales invariantes en el tiempo

Un sistema es LTI cuando satisface las 2 condiciones anteriores, de linealidad y de invariancia en el tiempo.



*** LTI ***

En este curso, solo estudiaremos sistemas lineales invariantes en el tiempo.

Sistemas

Fidelidad senoidal

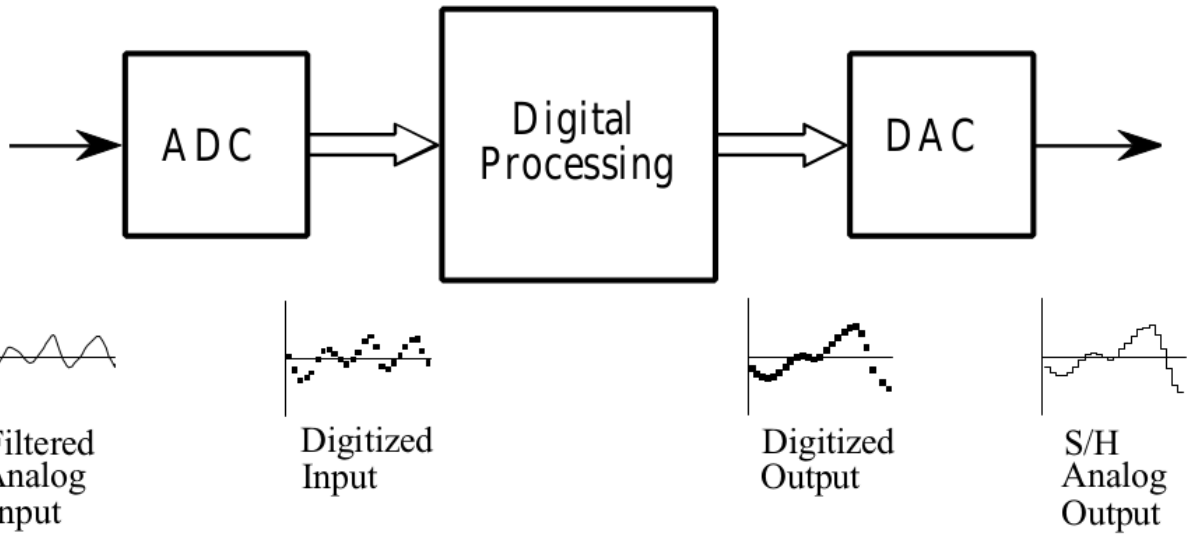
En todo sistema LTI para una entrada senoidal la salida es siempre senoidal.

Linealidad estática

En todo sistema LTI para una entrada constante (DC) la salida es siempre la entrada multiplicada por una constante.

ADC

Bloque incompleto de procesamiento



Que falta?

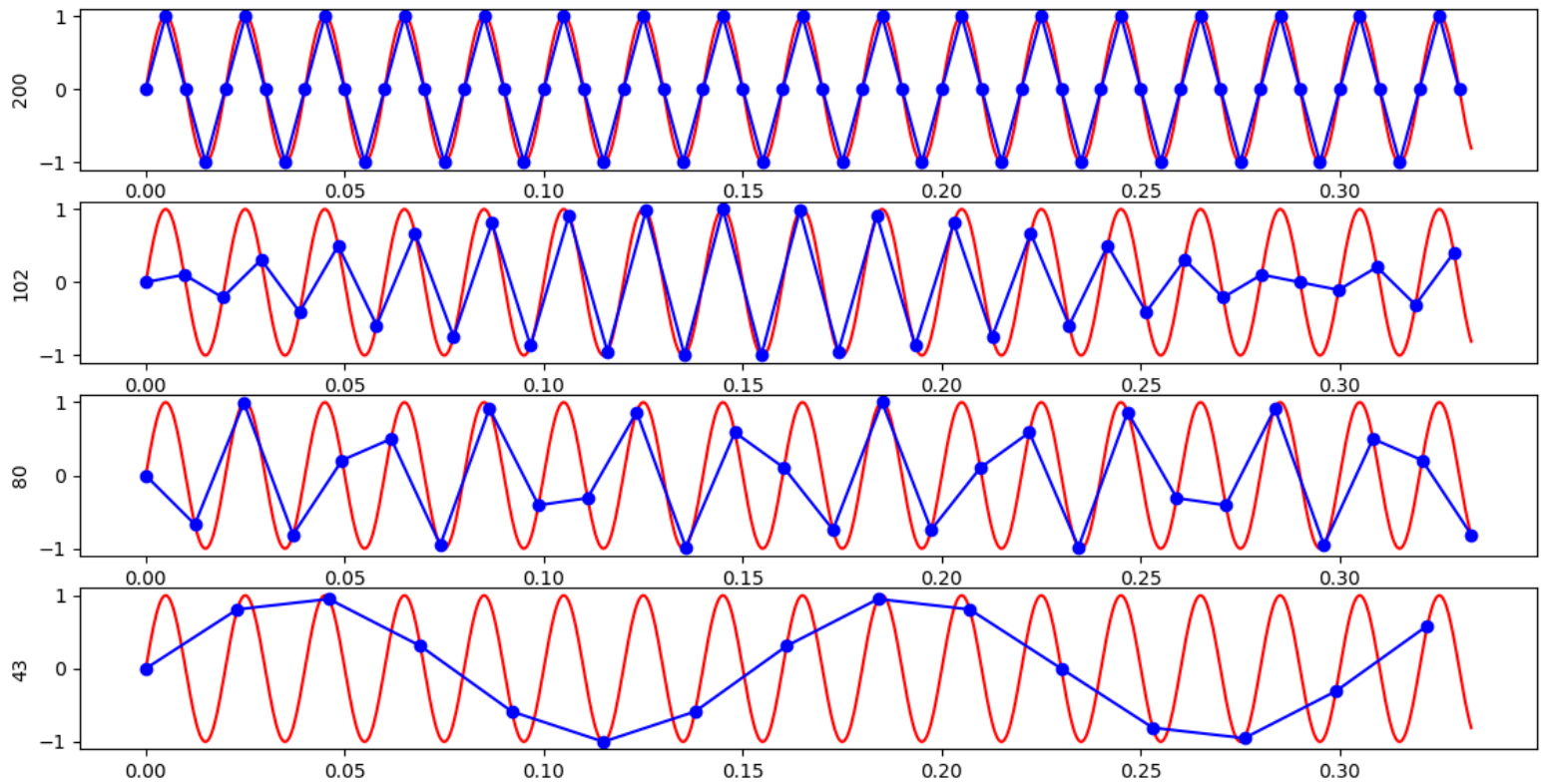
Aliasing

Disco Giratorio

Aliasing

Simulando en Python

Diferentes frecuencias de sampleo para capturar una señal de 50hz



Aliasing

Simulando en Python

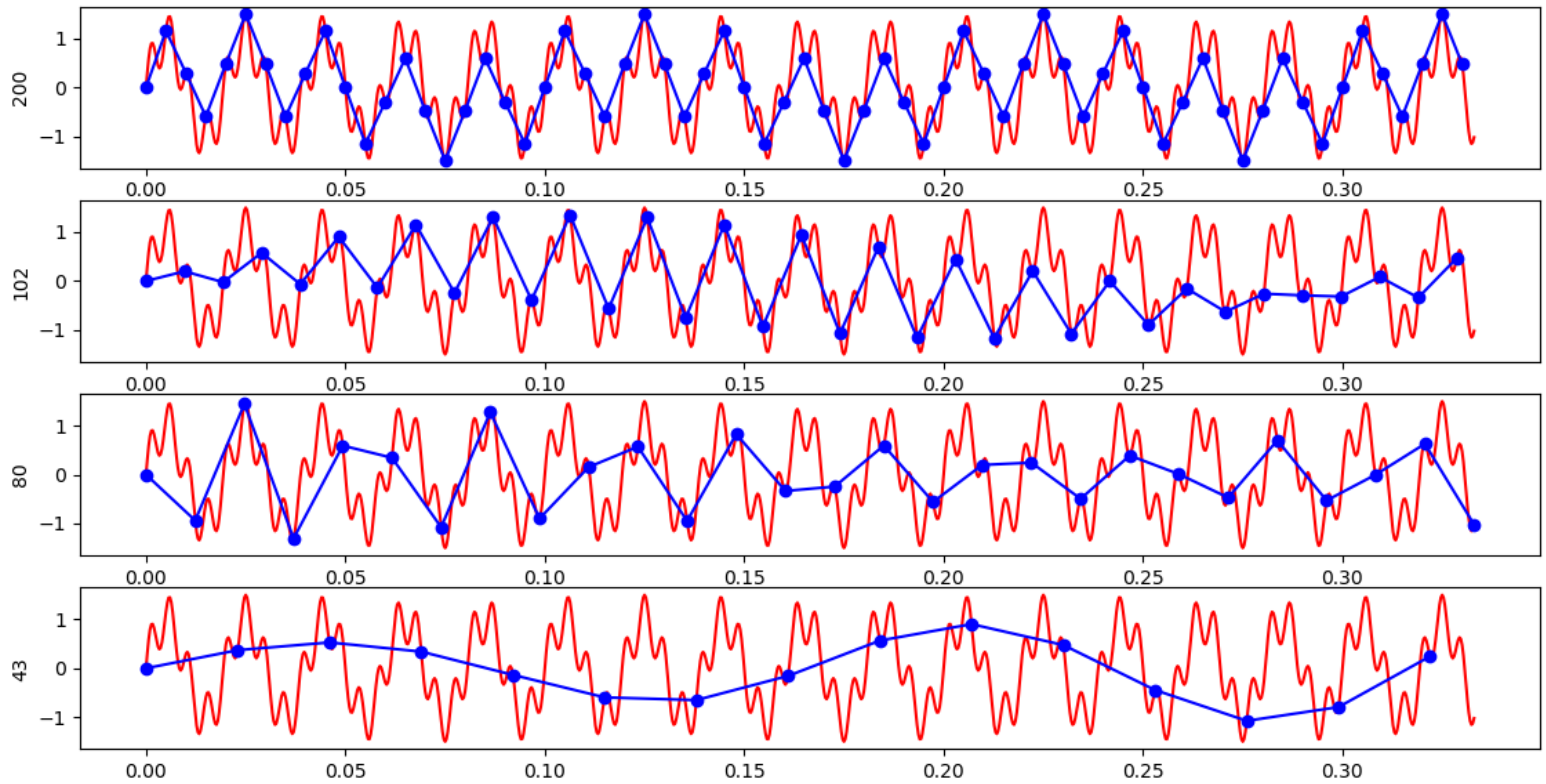
Diferentes frecuencias de sampleo para capturar una señal de 50hz

```
import numpy as np
import matplotlib.pyplot as plt

signalFrec = 50
NC         = 1000
fsC        = 3000
tC         = np.arange(0,NC/fsC,1/fsC)
signalC    = np.sin(2*np.pi*signalFrec*tC)
fsD        = [200,102,80,43]

fig      = plt.figure()
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()
```

Aliasing

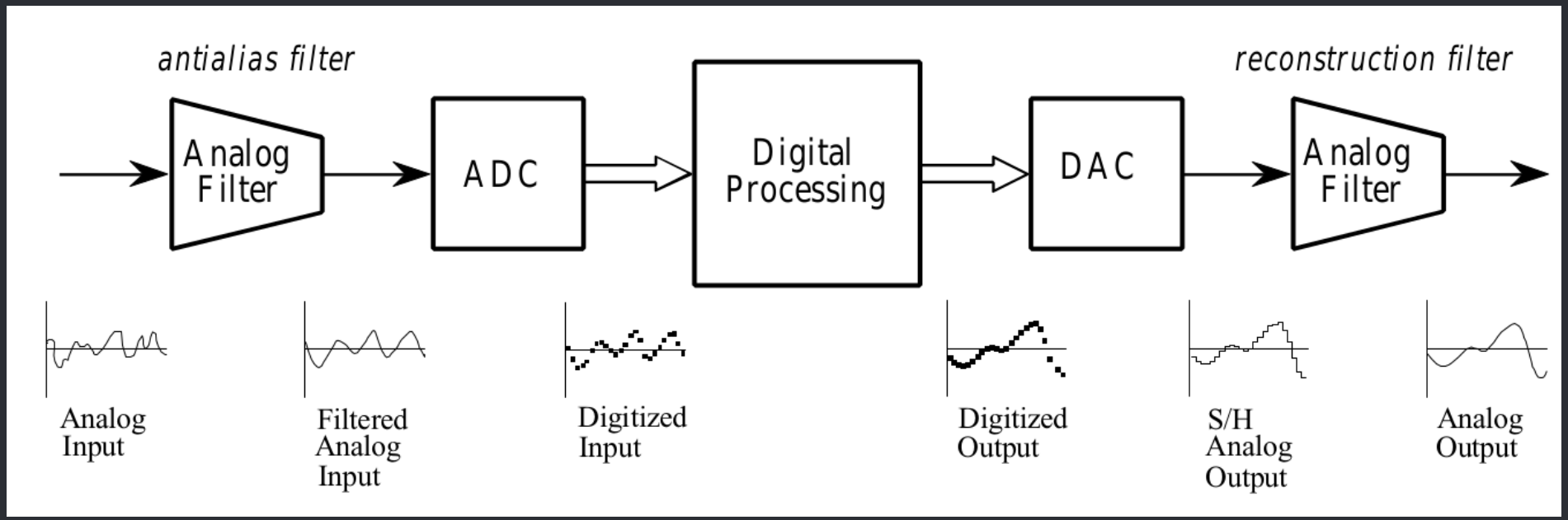
Simulando en Python

Que pasa si se suma ruido de alta frecuencia?



ADC

Bloque genérico de procesamiento



Agregamos el filtro antialising

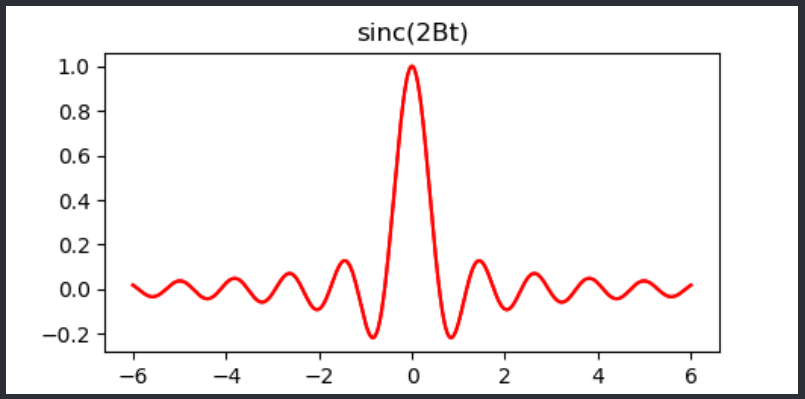
Teorema de sampleo

Teorema de Shannon

Teorema

La reconstrucción exacta de una señal periódica continua en banda base a partir de sus muestras, es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es superior al doble de su ancho de banda

$$x(t) = \sum_{n=-\infty}^{\infty} x_n \frac{\sin \pi(2Bt - n)}{\pi(2Bt - n)}.$$



Teorema de sampleo

Teorema de Shannon

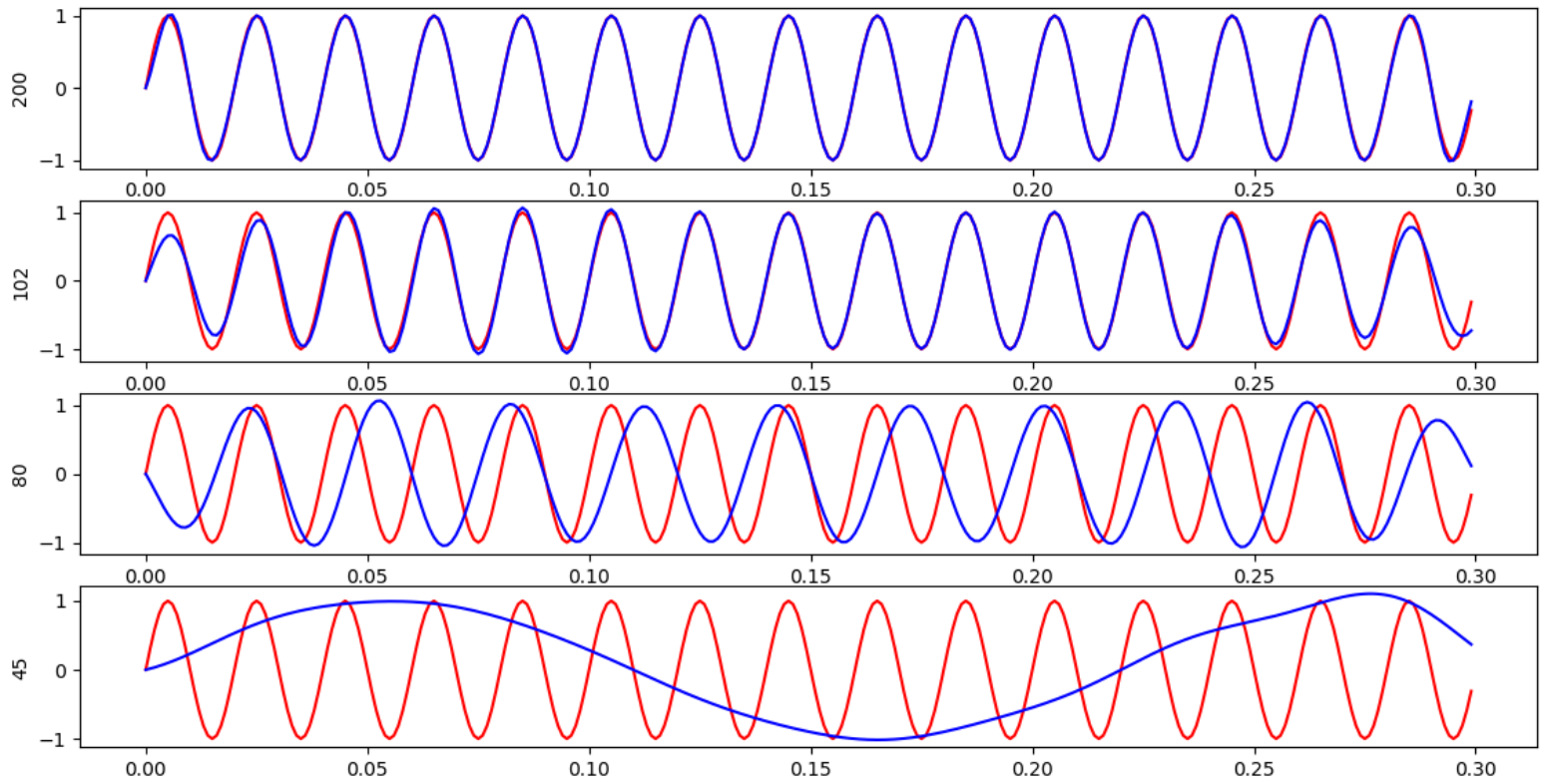
Sampleo e interpolado

```
#!/usr/bin/ip3
import numpy as np
import matplotlib.pyplot as plt

signalFrec = 50
NC = 300
fsC = 1000
tC = np.arange(0,NC/fsC,1/fsC)
signalC = np.sin(2*np.pi*signalFrec*tC)
#signalC = np.sin(2*np.pi*signalFrec*tC)+0.5*np.sin(2*np.pi*210*tC)
fsD = np.array([200,102,80,45])
fig = plt.figure()

def interpolate(x, s, u):
    y=[]
    B = 1/(2*(s[1] - s[0]))
    for t in u:
        prom=0
        for n in range(len(x)):
           prom+=x[n]*np.sinc(2*B*t-n)
        y.append(prom)
    return y

for i in range(len(fsD)):
    contiAxe = fig.add_subplot(4,1,i+1)
    Xt=interpolate(signalC[::fsC//fsD[i]],tC[::fsC//fsD[i]],tC)
    plt.plot(tC,signalC,'r-',tC,Xt,'b-')
    contiAxe.set_ylabel(fsD[i])

plt.show()
```

Teorema de sampleo

Teorema de Shannon

Sampleo e interpolado

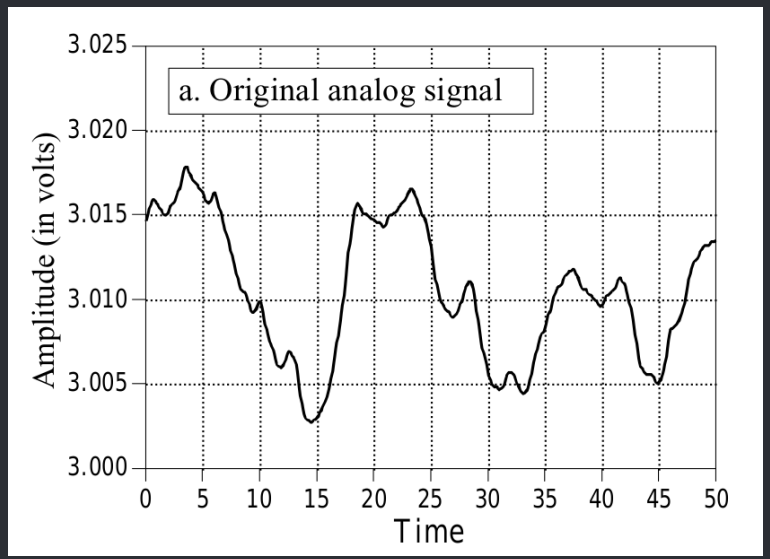


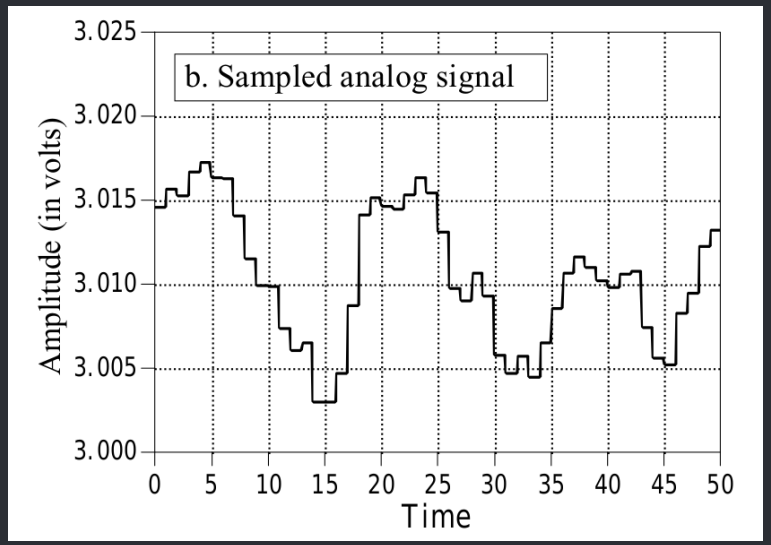
Sampleo

Filtro Antialias

FAA

Filtro analógico Pasabajos que elimina o al menos mitiga el efecto de aliasing



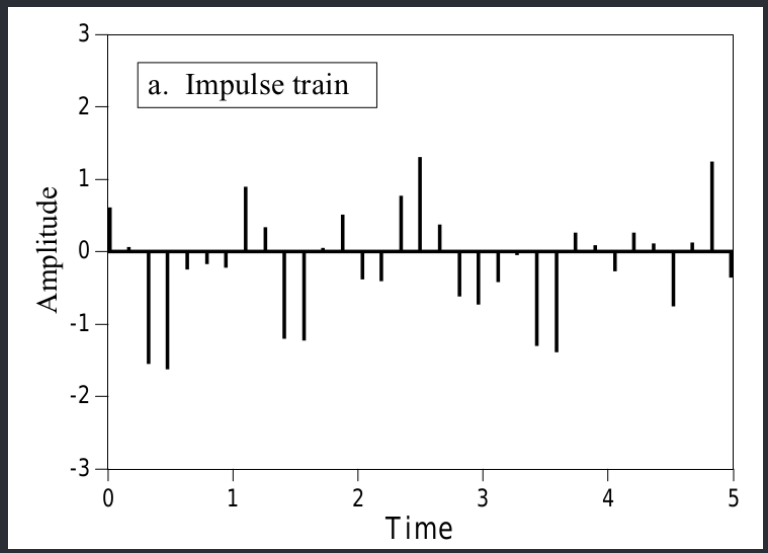


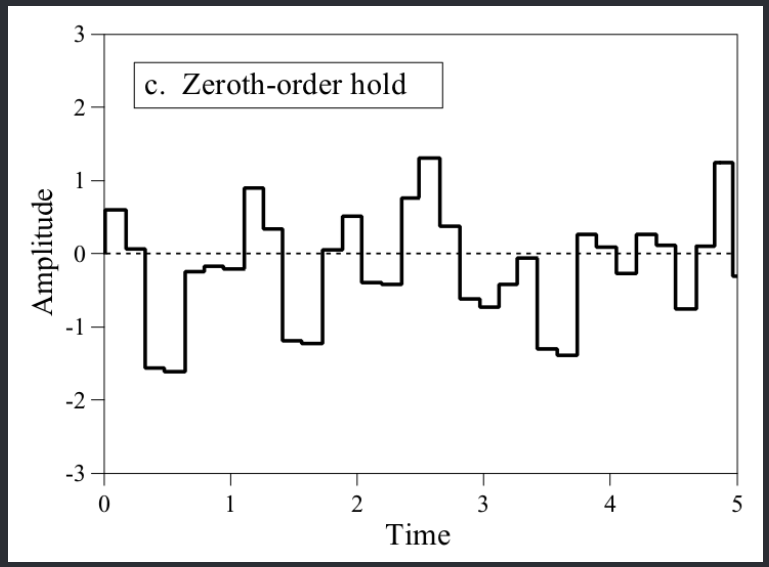
Sampleo

Filtro reconstructor

Filtro reconstructor

Filtro analógico Pasabajos que suaviza la salida del DAC eliminando frecuencias mas alla de la Fs/2



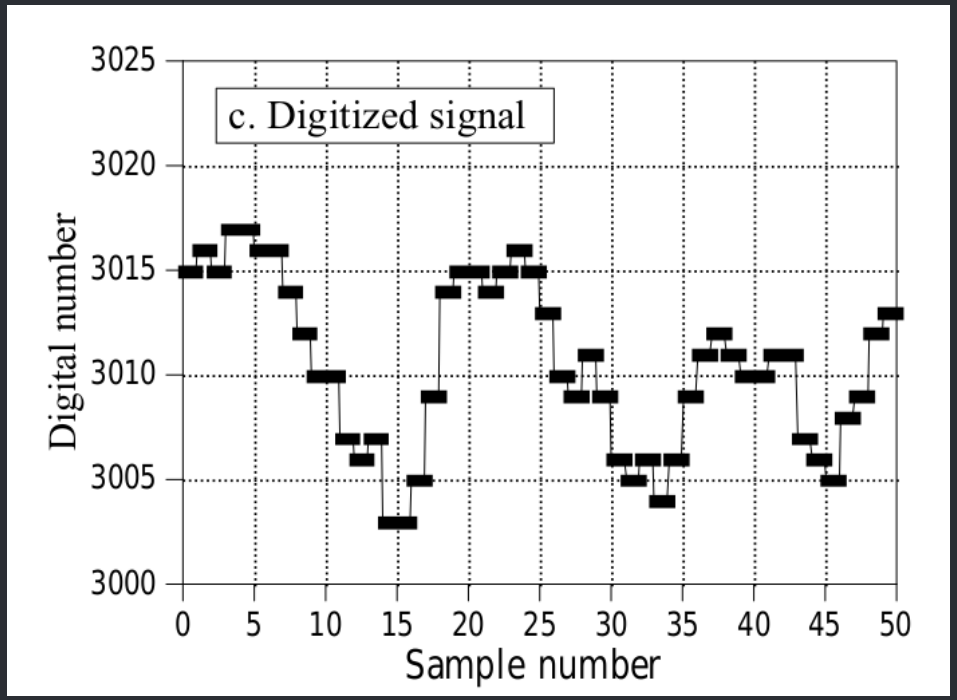


Sampleo

Digitado

Digitado o cuantizado

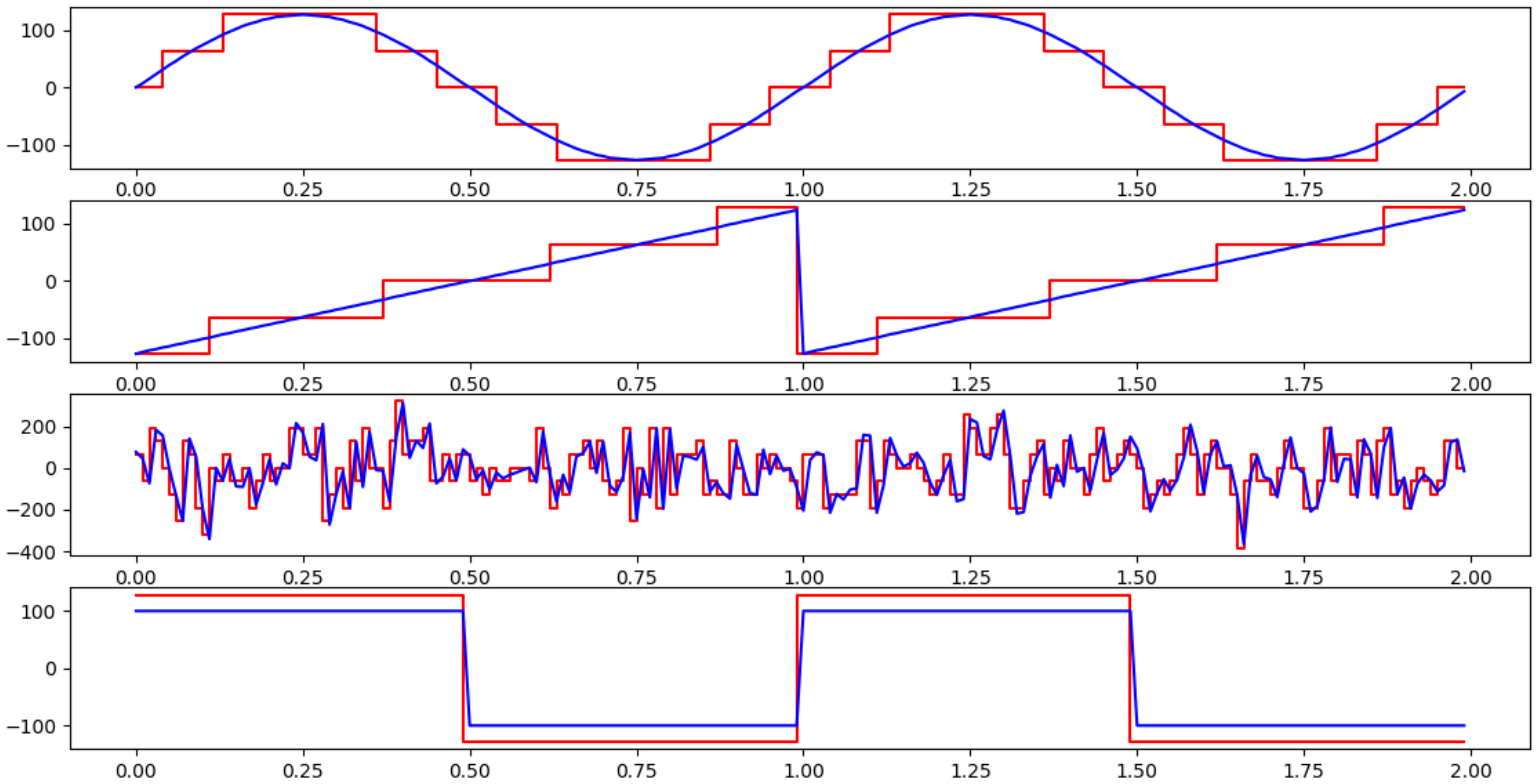
Proceso de asignar un patron de bits a una muestra



Ruido de cuantización

Ejemplo de cuantización

Diferentes formas de onda cuantizadas



Ruido de cuantización

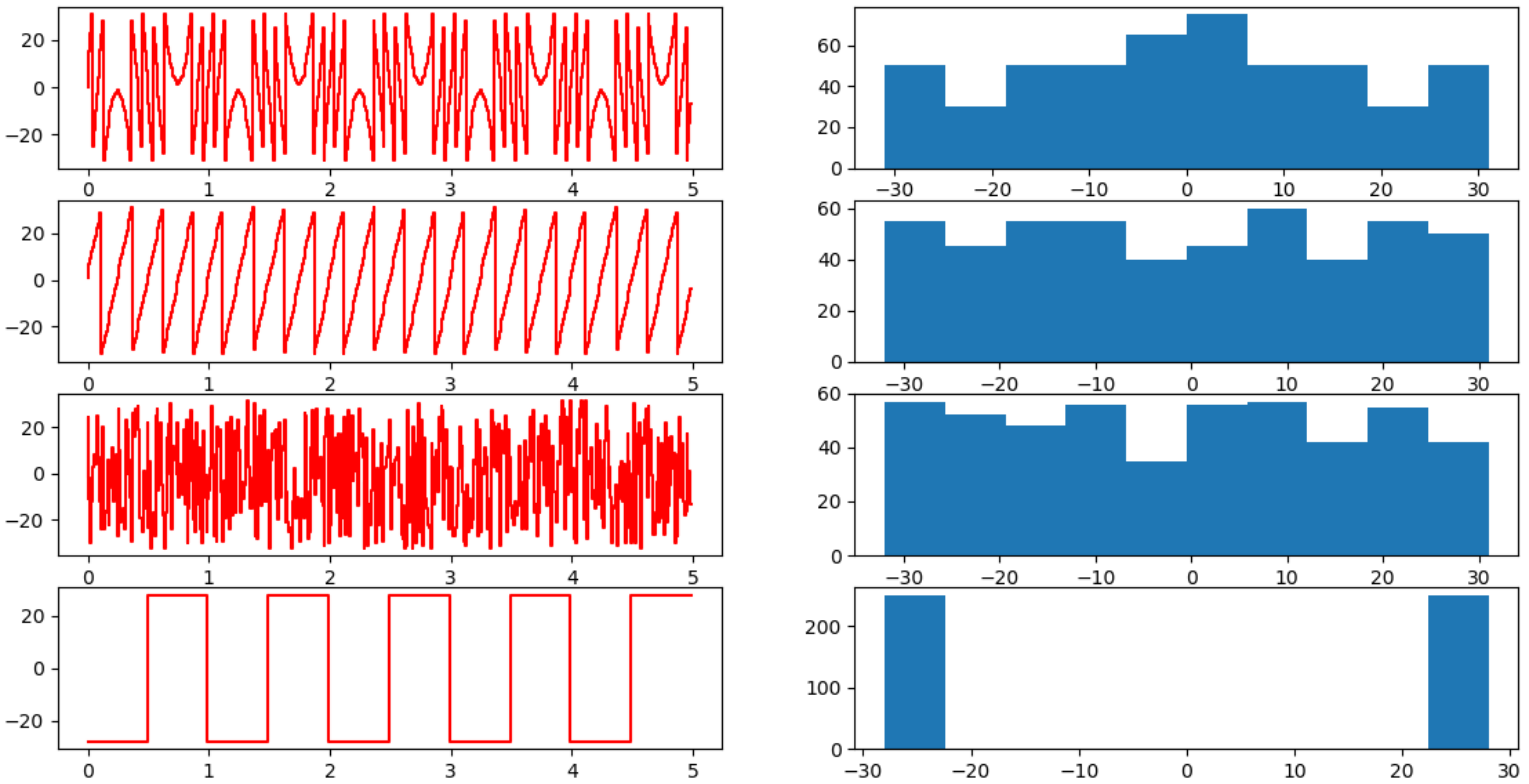
Cuantización en python

```
import numpy as np
import scipy.signal as sc
import matplotlib.pyplot as plt

signalFrec = 1
NC         = 200
fsC        = 100
Bits       = 2
tC         = np.arange(0,NC/fsC,1/fsC)
signalC    = np.array([(2**7-1)*np.sin(2*np.pi*signalFrec*tC),
             (2**7-1)*sc.sawtooth(2*np.pi*tC,1),
             (2**7-1)*np.random.normal(0,1,len(tC)),
             100*sc.square(2*np.pi*tC,0.5)],dtype='int16')

signalQ  = np.copy(signalC)
signalQ += (2**(8-Bits))//2
signalQ  &= 0xFFFF<<(8-Bits)

fig      = plt.figure()
for i in range(len(signalC)):
    contiAxe = fig.add_subplot(4,1,i+1)
    plt.step(tC,signalQ[i],'r-')
    plt.plot(tC,signalC[i],'b-')

plt.show()
```

Ruido de cuantización

Histogramas

Histogramas de ruido para cada señal



Ruido de cuantización

Histogramas

Histogramas en Python

```
import numpy as np
import scipy.signal as sc
import matplotlib.pyplot as plt

signalFrec = 1
NC         = 500
fsC        = 100
Bits       = 2
tC         = np.arange(0,NC/fsC,1/fsC)
signalC    = np.array([(2**7-1)*np.sin(2*np.pi*signalFrec*tC),
             (2**7-1)*sc.sawtooth(2*np.pi*tC,1),
             (2**7-1)*np.random.normal(0,1,len(tC)),
             100*sc.square(2*np.pi*tC,0.5)],dtype='int16')

signalQ  = np.copy(signalC)
signalQ += (2**(8-Bits))//2
signalQ  &= 0xFFFF<<(8-Bits)

fig      = plt.figure()
for i in range(len(signalC)):
    contiAxe = fig.add_subplot(4,2,2*i+1)
    plt.step(tC,signalC[i]-signalQ[i],'r-')
    contiAxe = fig.add_subplot(4,2,2*i+2)
    plt.hist(signalC[i]-signalQ[i])

plt.show()
```

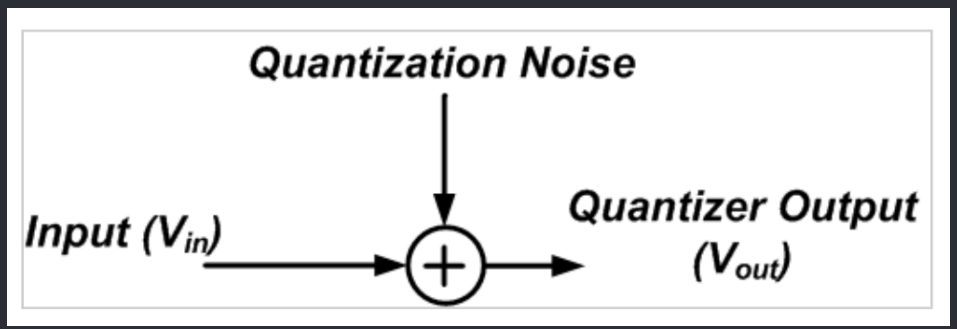
Ruido de cuantización

Modelo estadístico

En el caso de que se cumplan las siguientes premisas:

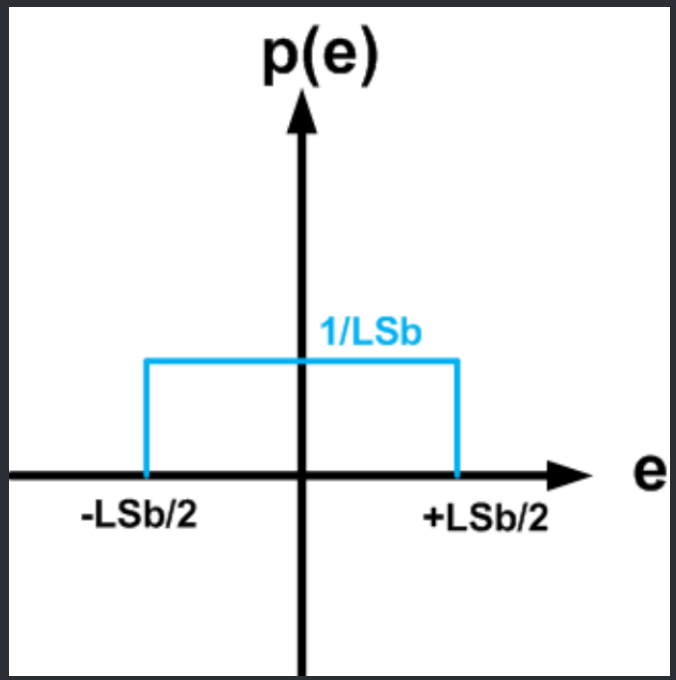
- La entrada se distancia de los diferentes niveles de cuantización con igual probabilidad
- El error de cuantización NO esta correlacionado con la entrada
- El cuantizador cuanta con un numero relativamente largo de niveles
- Los niveles de cuantización son uniformes

Se puede considerar la cuantización como un ruido aditivo a la señal según el siguiente esquema:



Ruido de cuantización

Función densidad de probabilidad



$$\int_{-\frac{lsb}{2}}^{\frac{lsb}{2}} p(e)de = 1$$

Ruido de cuantización

Potencia de ruido de cuantización

$$P_q = \int_{-\frac{lsb}{2}}^{\frac{lsb}{2}} e^2 p(e) de$$

$$P_q = \int_{-\frac{lsb}{2}}^{\frac{lsb}{2}} e^2 \frac{1}{lsb} de$$

$$P_q = \frac{1}{lsb}\left(\frac{e^3}{3}\Big|_{-\frac{lsb}{2}}^{\frac{lsb}{2}} \right)$$

$$P_q = \frac{1}{lsb}\left(\frac{(\frac{lsb}{2})^3}{3} - \frac{(\frac{-lsb}{2})^3}{3} \right)$$

$$P_q = \frac{1}{lsb}\left(\frac{lsb^3}{24} + \frac{lsb^3}{24} \right)$$

Potencia de ruido de cuantización

$$P_q = \frac{lsb^2}{12} \tag{1}$$

Ruido de cuantización

Relación señal a ruido

$$input = \frac{Amp}{2}\sin(t)$$

$$P_{input} = \frac{1}{T}\int_0^T \left(\frac{Amp}{2}\sin(t)\right)^2 dt$$

$$P_{input} = \frac{1}{T}\left(\frac{Amp}{2}\right)^2 * \left(\frac{t}{2} - \frac{\sin(2t)}{4}\right)\Big|_0^T$$

$$P_{input} = \frac{Amp^2}{4T}\frac{T}{2}$$

$$P_{input} = \frac{Amp^2}{8}$$

$$lsb = \frac{Amp}{2^N}$$

$$P_{ruido} = \frac{lsb^2}{12}$$

$$P_{ruido} = \frac{\left(\frac{Amp}{2^N}\right)^2}{12}$$

$$P_{ruido} = \frac{Amp^2}{12 * 2^{2N}}$$

Ruido de cuantización

Relación señal a ruido

$$SNR = 10 \log_{10} \left(\frac{P_{input}}{P_{ruido}} \right)$$

$$SNR = 10 \log_{10} \left(\frac{\frac{Amp^2}{8}}{\frac{Amp^2}{12 * 2^{2N}}} \right)$$

$$SNR = 10 \log_{10} \left(\frac{3 * 2^{2N}}{2} \right)$$

$$SNR = 10 \log_{10} \left(\frac{3}{2} \right) - 10 \log_{10} \left(2^{2N} \right)$$

SNR

$$SNR = 1{,}76 + 6{,}02 * N \tag{2}$$

$$SNR_{N=10} \approx 60dB$$

$$SNR_{N=11} \approx 66dB$$

Ruido de cuantización

Densidad espectral de potencia de ruido

Si consideramos la potencia de ruido uniformemente distribuido en todo el espectro desde $-Fs$ hasta $+Fs$, nos queda que:

Densidad espectral de potencia de ruido

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

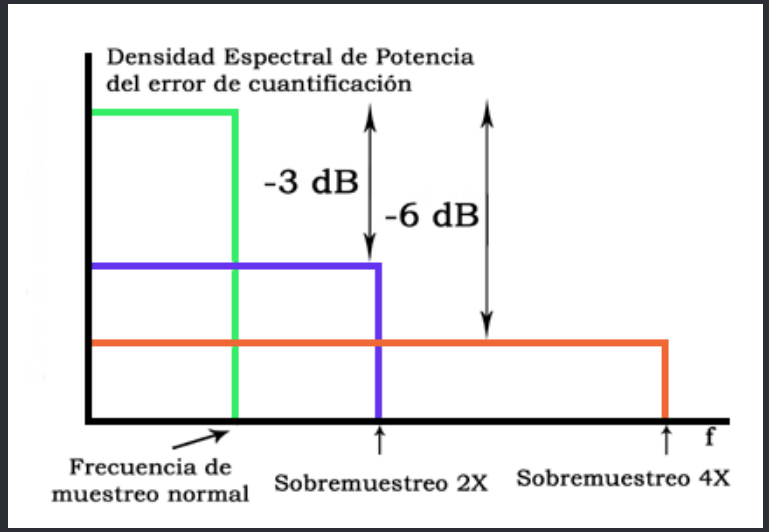
Entonces como puedo mejorar la SNR de un sistema?

Sobremuestreo

Densidad espectral de potencia de ruido

Oversampling x4

$$S_{espectral}(f) = \frac{P_q}{4 * Fs}$$



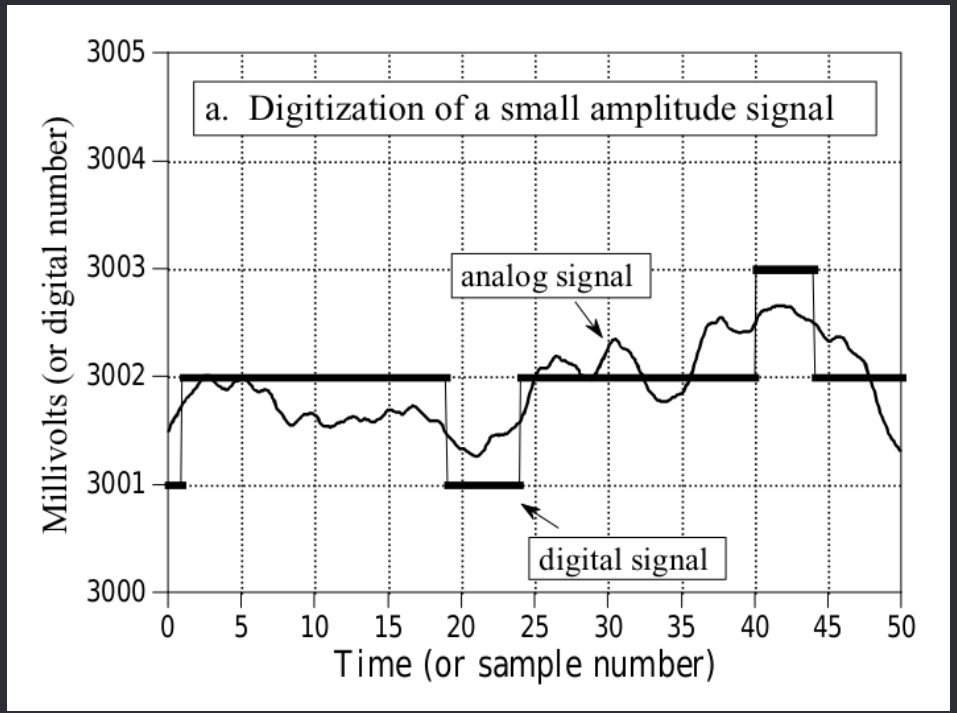
Que hago si tengo un AD de 10bits y deseo una SNR de 72dB? $SNR_{10} \approx 66dB$

Pero si sobremuestreo a 4x obtengo 6dB extras

Dithering

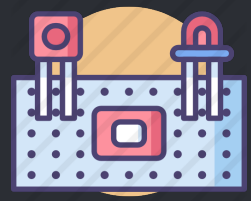
Dithering

Tecnica de agregado de ruido antes del ADC para prevenir que señales con poca variacion sean samoleadas siempre con el mismo valor

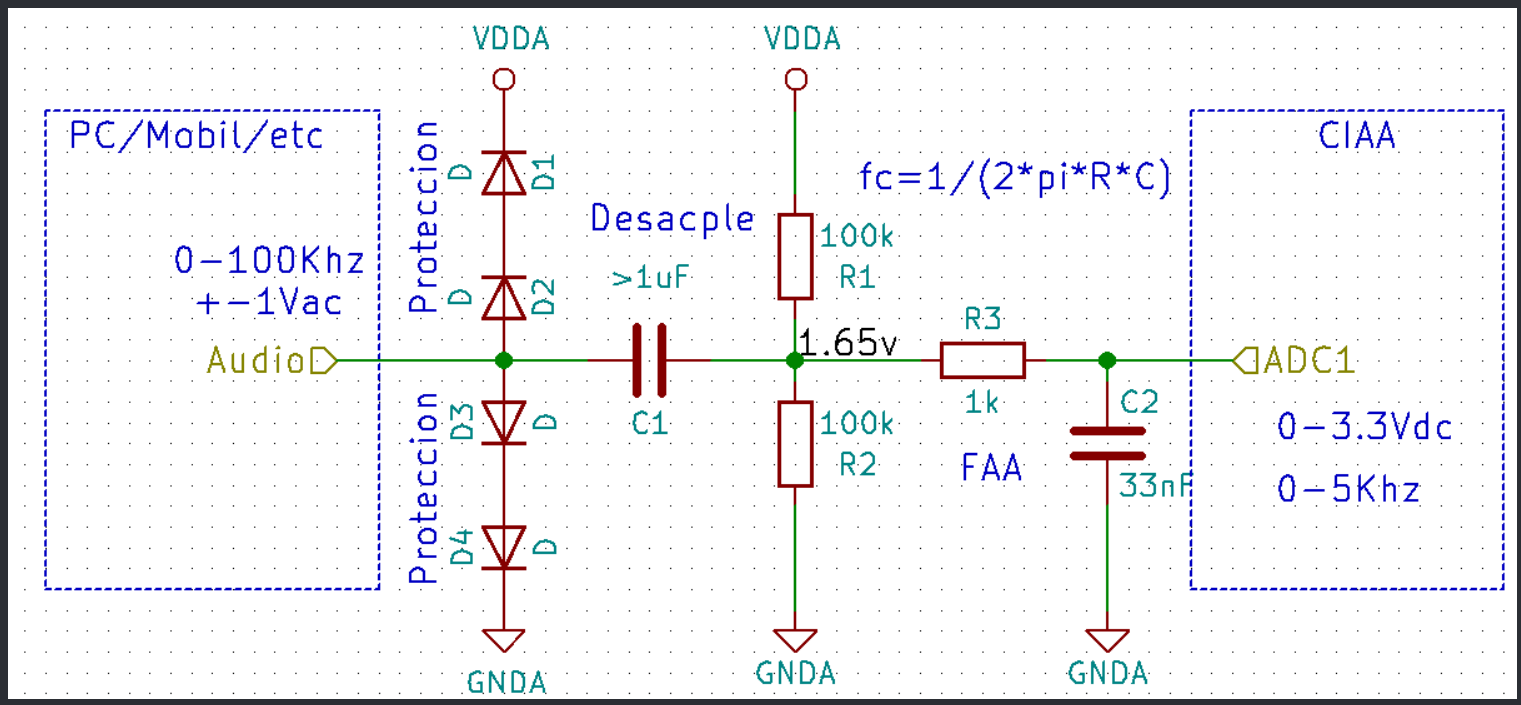


Sampleo

Propuesta para acondicionamiento de señal

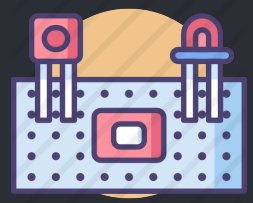


Acondicionar la señal de salida del dispositivo de sonido (en PC ronda ±1V) al rango del ADC del hardware. En el caso de la CIAA sera de 0-3.3V.
Se propone el siguiente circuito, que minimiza los componentes sacrificando calidad y agrega en filtro anti alias de 1er orden.



Sampleo

Propuesta para acondicionamiento de señal

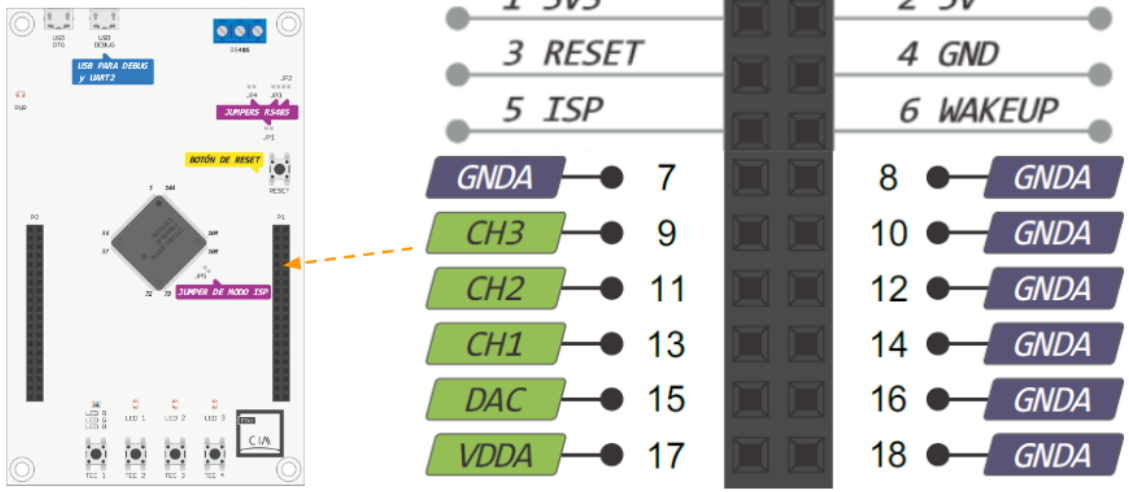


Pinout de la CIAA para conectar el ADC/DAC

ADC y DAC en la EDU-CIAA-NXP

Mapeo de ADC y DAC en la biblioteca sAPI:

- 3 entradas analógicas nombradas CH1, CH2 y CH3 (ADC).
- 1 salida analógica nombrada DAC.

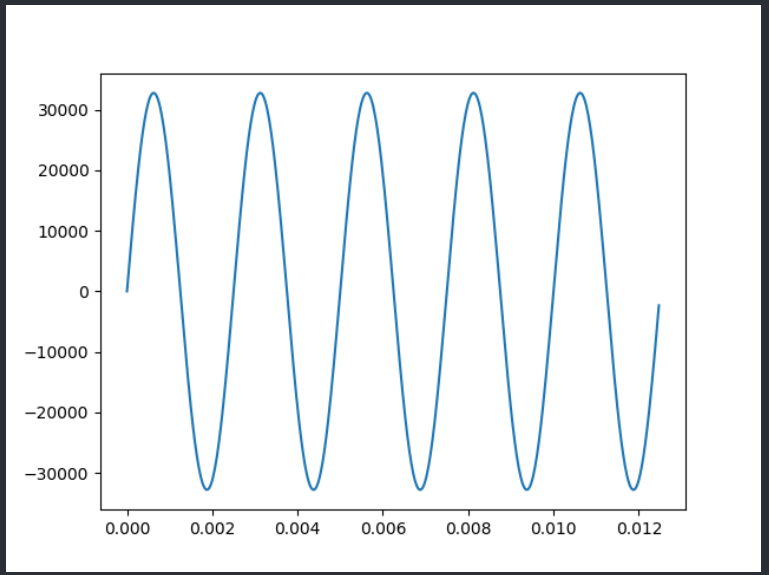


Generación de audio con Python

simpleaudio lib

para instalar pulseaudio: https://simpleaudio.readthedocs.io/en/latest/installation.html Como herramienta para la generación de audio para capturar con la CIAA se propone el uso de simpleaudio con el siguiente template de código como base

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as sc
import simpleaudio as sa

f          = 200
fs         = 44100
sec        = 10
t = np.arange(0,sec,1/fs)
note = (2**15-1)*np.sin(2 * np.pi * f * t)
#jnote = (2**15-1)*sc.sawtooth(2 * np.pi * f * t)
#jnote = (2**15-1)*sc.square(2 * np.pi * f * t)

fir=plt.figure(1)
plt.plot(t[0:5*fs//f],note[:5*fs//f])
plt.show()

audio = note.astype(np.int16)
for i in range(100):
    play_obj = sa.play_buffer(audio, 1, 2, fs)
    play_obj.wait_done()
```

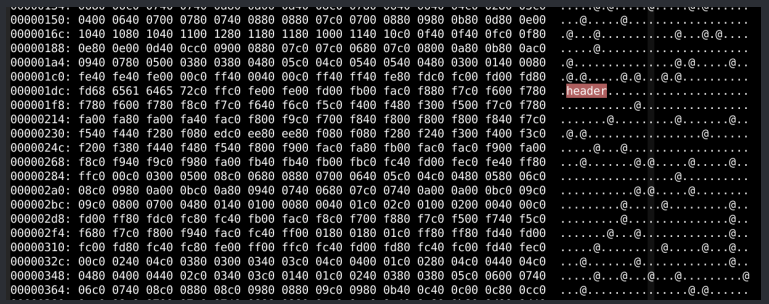


Captura de audio con la CIAA

Ciaa->Uart->picocom->log.bin

Se detalla código simple de sampleo y envío por UART

```
#include "sapi.h"

#define LENGTH 512
int16_t adc [ LENGTH ];
uint16_t sample = 0;

int main ( void ) {
   boardConfig ( );
   uartConfig ( UART_USB, 460800 );
   adcConfig ( ADC_ENABLE );
   cyclesCounterInit ( EDU_CIAA_NXP_CLOCK_SPEED );
   while(1) {
      cyclesCounterReset();
      uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[
            sample] ,sizeof(adc[0]) );
      adc[sample] = ((int16_t )adcRead(CH1)-512)<<6;
      if ( ++sample==LENGTH ) {
         sample = 0;
         uartWriteByteArray ( UART_USB ,"header" ,6 );
         gpioToggle( LEDR);
      }
   }
   while(cyclesCounterRead()< 204000)
      ;
}
```



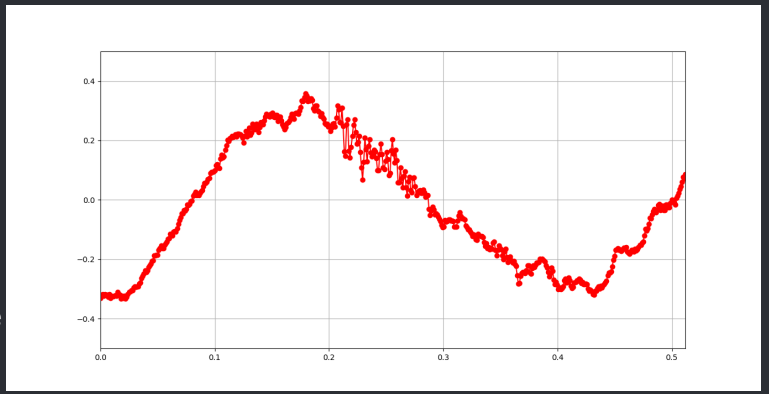
picocom /dev/ttyUSB1 -b 460800 –logfile=log.bin

Captura de audio con la CIAA

Uart->Python

Se detalla la lectura de un log y visualización en tiempo real de los datos

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

length = 512
fs     = 1000
header = b'header'
fig    = plt.figure()
adcAxe = fig.add_subplot ( 1,1,1 )
adcLn, = plt.plot ( [],[],'r-o'   )
adcAxe.grid       ( True          )
adcAxe.set_ylim   ( -0.5 ,0.5     )
adcAxe.set_xlim   ( 0 , length/fs )

def initFiles():
    global logFile
    logFile   = open("log.bin","rb")
    logFile.seek(0, os.SEEK_END)

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(length):
        adc.append (readInt4File(logFile
                  )/(2**15))
    time = np.linspace(0,length/fs,
           length)
    adcLn.set_data ( time,adc )
    return adcLn,

initFiles()
ani=FuncAnimation(fig,update,10,None,
       blit=True,interval=10,repeat=
       True)
plt.show()
```



Sistemas de números

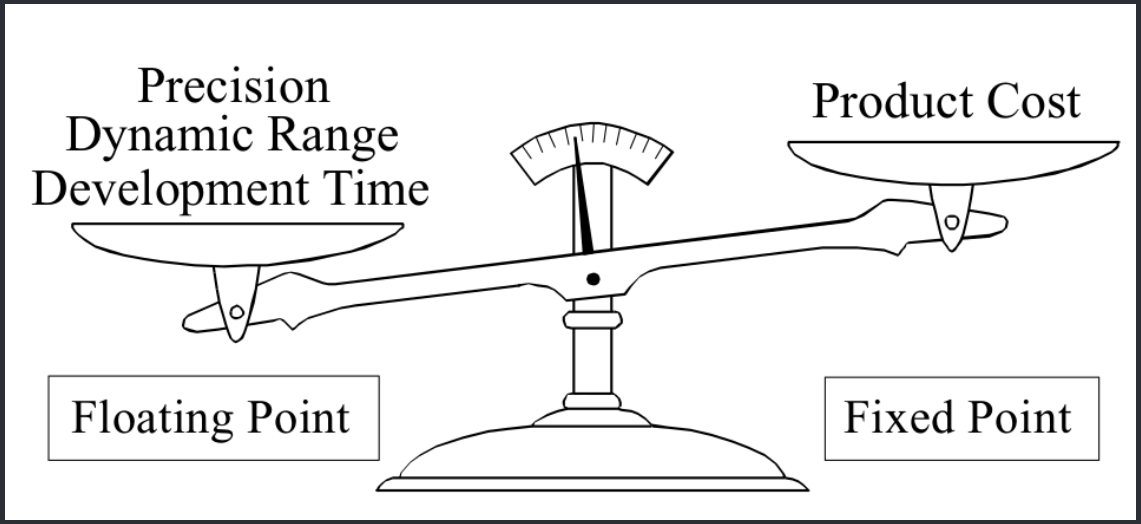
Punto fijo vs punto flotante

Punto fijo:

- Cantidad de patrones de bits= 65536
- Gap entre números constante
- Rango dinámico 32767, —32768
- Gap 10 mil veces mas chico que el numero

Punto flotante:

- Cantidad de patrones de bits= 4,294,967,296
- Gap entre números variable
- Rango dinámico $\pm 3,4e10^{38}$, $\pm 1,2e10^{-38}$
- Gap 10 millones de veces mas chico que el numero



Sistemas de números

Sistema Q

Qm.n:

- m: cantidad de bits para la parte entera
- n: cantidad de bits para la parte decimal

Q1.15:

1000 0000 0000 0000 = -1

0111 1111 1111 1111 = $1/2 + 1/4 + 1/8 + .. + 1/2^{15} = 0,99$

Q2.14:

1010 0000 0000 0000 = -1.5

0101 0000 0000 0000 = 1.25