CAPÍTULO 1.
Del Mundo Análogo al Mundo Digital

En este capítulo encontrarás una breve introducción al procesamiento digital de señales, específicamente en relación con los conceptos de muestreo, cuantización y costo de almacenamiento/transmisión asociados al proceso de conversión análogo a digital (A/D).

Al finalizar el capítulo, deberás estar en capacidad de:

1. Explicar el concepto de muestreo de señales análogas/continuas.
2. Explicar el concepto de cuantización de muestras.
3. Seleccionar adecuadamente los parámetros de frecuencia de mues treo y bits de resolución en la conversión A/D, de acuerdo con el espectro de la señal y su comportamiento en el dominio del tiempo.
4. Explicar el costo de almacenamiento/transmisión del proceso de conversión A/D de señales continuas/análogas.
5. Explicar el efecto en frecuencia de muestrear una señal análoga/ continua.

El Procesamiento Digital de Señales es un conjunto de técnicas y métodos que permiten manipular una señal para obtener información de ella (patrones), o para modificarla o transformarla. Por ejemplo, la señal de voz es una señal análoga en tiempo continuo que contiene información de entonación, género del hablante, idioma, entre otros, que puede ser utilizada para identificar qué persona está pronunciando un mensaje o discurso. En este caso, el procesamiento digital de la señal se enfoca en identificar *patrones de voz* que permitan caracterizar al hablante, y compararlo con una base de datos previamente almacenada en el sistema. También, hoy en día encontramos dispositivos celulares que utilizan reconocimiento facial como medio para desbloquear el acceso al sistema, sustituyendo o reemplazando la opción clásica de clave numérica; por lo cual, el celular debe identificar *"características faciales"* que permitan corroborar si el rostro que está frente a la cámara es el autorizado para desbloquearlo.

Pero ¿cómo pasamos del mundo análogo/continuo al mundo digital/discreto? Gran parte de las señales que encontramos en la naturaleza son análogas (infinitos valores de amplitudes posibles) que se van actualizando a lo largo de la variable independiente, que típicamente es el tiempo (con infinitos valores de tiempo posibles), que deben ser transformadas antes de poder ser utilizadas por un sistema digital.

El proceso se conoce como conversión análogo-digital (ó A/D), el cual consiste en seleccionar un número finito de valores de tiempo en los que representaremos sus amplitudes en un número finito de bits. De tal forma que tanto la variable independiente (tiempo), como la variable dependiente (amplitud de la señal) son discretizados. En la Figura 1 encontrarás una gráfica ilustrativa de la conversión A/D.

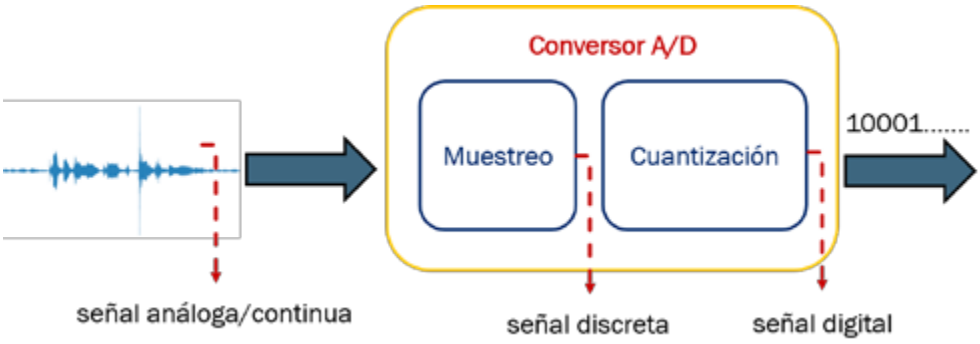

Figura 1. Diagrama general de un proceso de conversión A/D.

En las siguientes subsecciones encontrarás en detalle los conceptos de muestreo y cuantización, su costo de almacenamiento y transmisión, así como el efecto del muestreo en el espectro de la señal.

1.1. MUESTREO DE LA SEÑAL ANÁLOGA/CONTINUA

En la primera parte del proceso de conversión A/D, se selecciona un número de muestras por segundo de la señal, conocido como frecuencia de muestreo (f_s). De esta forma, si, por ejemplo, la señal tiene una duración de 10 segundos y la frecuencia de muestreo es de 8 kHz, entonces, la cantidad total de muestras es de 80.000. El valor de f_s, en el caso de muestreo equi-espaciado, debe satisfacer el criterio de Nyquist, el cual establece que:

$$f_s \geq 2 * f_{max} \qquad \text{Ecuación 1}$$

Donde f_{max} corresponde a la frecuencia máxima de la señal de tiempo continuo. Por ejemplo, si el espectro de nuestra señal tiene el comportamiento de la Figura 2, entonces la frecuencia de Nyquist es de 8 kHz. En otras palabras, una $f_s = 8\ kHz$ solo es adecuada para señales cuya $f_{max} = 4\ kHz$.

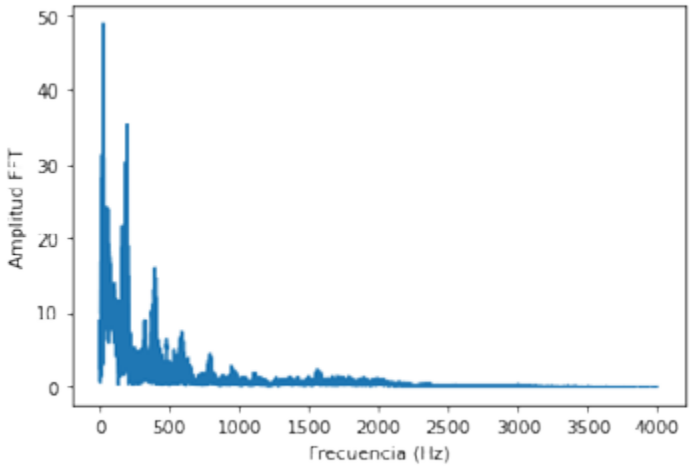

Figura 2. Ejemplo de espectro de señal de voz, f_s = 8 kHz.

La lectura y procesamiento de un archivo de voz (ej. en formato wav), está soportado en muchos lenguajes de programación. Para el caso del lenguaje Python, podemos utilizar la librería *Librosa* con el fin de cargar la señal en el entorno de ejecución (por ejemplo, en un *Jupyter* notebook como *CoLaboratory)*. Esta librería permitirá también visualizar la señal o conocer la f_s con la que fue muestreada.

Específicamente, en lenguaje Python escribimos el siguiente código para la visualización de la señal en el dominio del tiempo:

```
import librosa
import librosa.display
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
from scipy.io import wavfile
from scipy.fft import fftshift

plt.rcParams["figure.figsize"] = (14,5)
filename = 'audio.wav'

# Se debe asignar sr=None para que se conserve la fs original del audio.
# En caso contrario, se re-muestrea a 22050 Hz.
audio, fs = librosa.load(filename, sr=None)
librosa.display.waveplot(audio, sr=fs);

print("frecuencia de muestreo de la señal:", fs, "Hz")
print("cantidad de muestras de la señal:", len(audio))
```

Obteniendo como resultado:

```
frecuencia de muestreo de la señal: 8000 Hz
cantidad de muestras de la señal: 24000
```

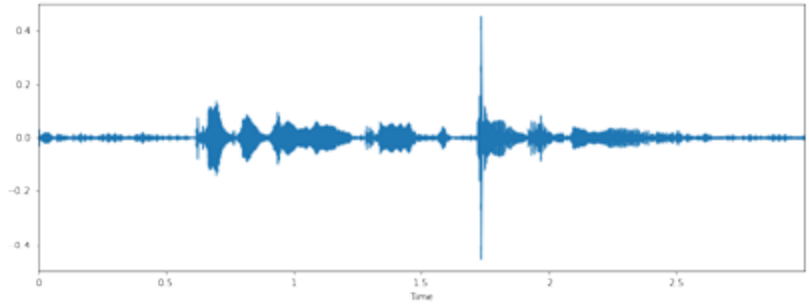

Figura 3. Ejemplo de señal de voz en el dominio del tiempo.

De acuerdo con la Figura 3, esta señal de voz tiene una duración de 3 segundos, y su amplitud se encuentra comprendida entre [-0.45 0.45], aproximadamente. Adicionalmente, en 1.7 segundos, se percibe un incremento significativo de la amplitud de la señal (tanto positiva como negativa) en relación con los demás valores de amplitud a lo largo de los 3 segundos. Teniendo en cuenta que la $f_s = 8\ kHz$, el total de muestras de la señal es de 24K.

Si queremos que el audio se ajuste al máximo volumen posible, podemos escalar su amplitud, así:

```
norm = max(np.absolute([min(audio), max(audio)]))
audio= audio /norm
librosa.display.waveplot(audio, sr=fs);
```

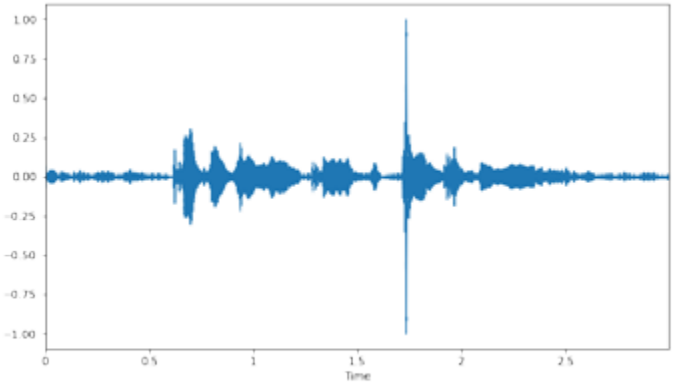

Figura 4. Ejemplo de señal de voz en el dominio del tiempo, con normalización de amplitud.

Esta nueva señal tiene una amplitud mayor a la señal original, y ahora se encuentra en el rango de [-1 1]. Adicionalmente, podemos reproducir el audio, con el siguiente código:

```
IPython.display.Audio(audio, rate=fs)
```

El cual genera un botón de reproducción

Posteriormente, es posible graficar el espectro de la señal con el siguiente código en Python:

```
import scipy.fftpack as  fourier

L=len(audio)
transformada = fourier.fft(audio)
magnitud = abs(transformada)
magnitud_lateral = magnitud[0:L//2]
fase = np.angle(transformada)
frecuencias = fs*np.arange(0, L//2)/L

plt.plot(frecuencias, magnitud_lateral)
plt.xlabel('Frecuencia (Hz)', fontsize='10')
plt.ylabel('Amplitud FFT', fontsize='10')
plt.show()
```

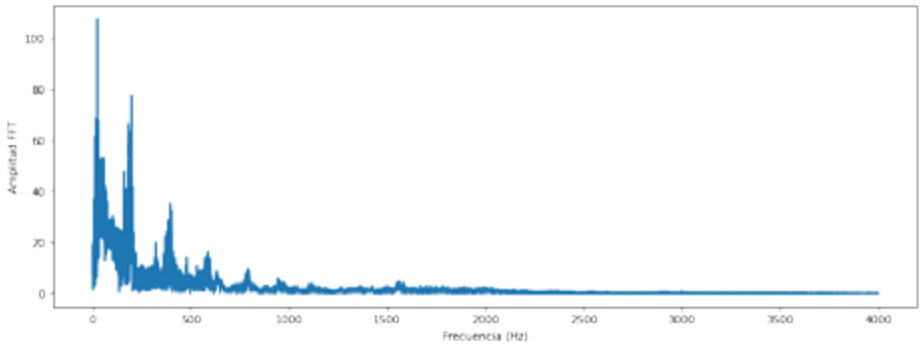

Figura 5. Espectro de la señal de voz de la Figura 4.

Pero ¿cómo sabemos si la frecuencia de muestreo de la señal en el proceso de conversión A/D fue adecuada? La respuesta la obtenemos en su espectro. Por ejemplo, para nuestro caso, las amplitudes de la FFT para frecuencias mayores de $2\ kHz$ son muy cercanas a cero y distan significativamente de las amplitudes en frecuencias inferiores a $1\ kHz$. De tal forma que, la mayor parte de la energía de la señal se encuentra en las frecuencias menores a 1kHz, y entonces $f_s = 8\ kHz$ es adecuada. Si por el contrario, en frecuencias cercanas a $4\ kHz$ las amplitudes de la FFT fuesen

comparativamente altas en relación con frecuencias menores, muy posiblemente la f_s seleccionada sería incorrecta, y tendríamos que escoger un valor mayor.

Supongamos que nuestra señal corresponde a un fragmento de música de un concierto de violín (Figura 6), cuyo espectro se presenta en la Figura 7.

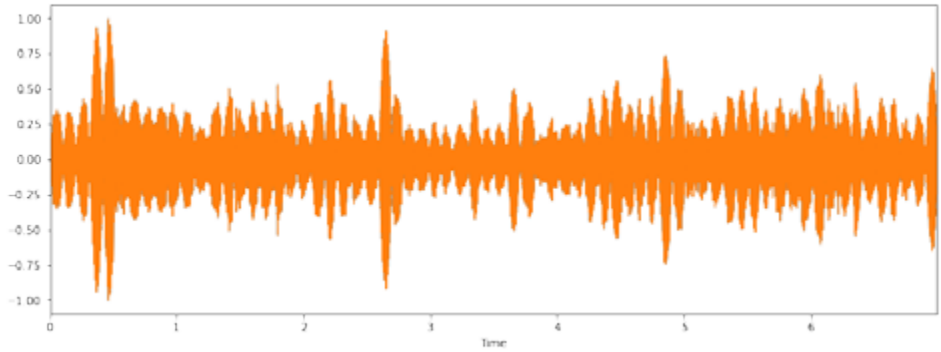

Figura 6. Ejemplo de señal de música en el dominio del tiempo.

A diferencia de la señal de voz, las amplitudes de la FFT cercanas a $4\ kHz$ no son significativamente pequeñas en relación con las amplitudes en frecuencias menores a $1\ kHz$, por lo que utilizar una $f_s > 8\ kHz$ es necesario, por ejemplo $f_s = 22\ kHz$.

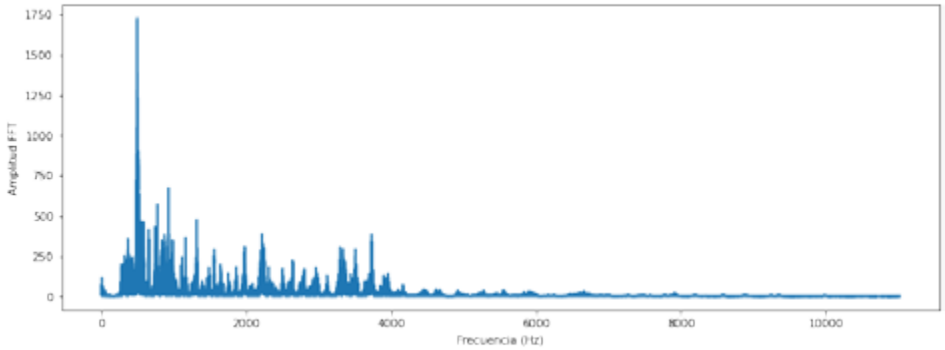

Figura 7. Espectro de la señal de música de la Figura 6.

Hasta aquí, hemos comprendido que no todas las señales necesitan la misma frecuencia de muestreo, y que a medida que la frecuencia máxima de la señal continua es mayor, debemos muestrear la señal con un número mayor de muestras por segundo. ¿Pero qué ocurriría si seleccionamos una f_s no adecuada, es decir que no cumpla el criterio de Nyquist? La respuesta se ilustrará a través de ejemplos.

Supongamos entonces que la señal de voz de la Figura 4 la re-muestreamos a $1\ kHz$, es decir, solamente conservaremos las componentes de frecuencias de los 0 Hz hasta los 500 Hz, como se presenta en la Figura 8.

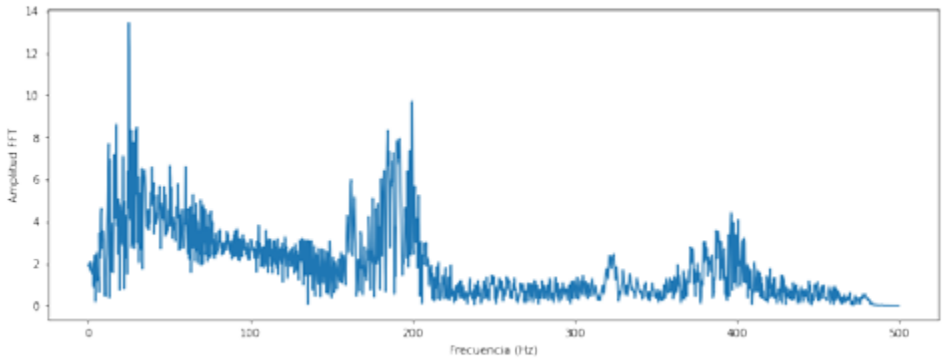

Figura 8. Espectro de la señal de voz de la Figura 4, re-muestreada a 1 kHz.

Esta nueva señal tiene el efecto de escucharse la voz ahogada, dado que, no cuenta con componentes de frecuencias altas, relacionadas con el detalle de la señal. Este fenómeno, el cual ocurre cuando la frecuencia de muestreo no es al menos el doble de la frecuencia máxima de la señal, se conoce como *aliasing*.

Finalmente, es necesario aclarar que, aunque en un dispositivo digital como un PC podemos graficar una señal con *apariencia de continua/análoga,* estas señales son en realidad *discretas/digitales*. Internamente, se realiza un proceso de interpolación que permite unir las amplitudes discretas para que luzcan como una señal que varía para valores infinitos de tiempo.

Específicamente en lenguaje Python, se puede utilizar la librería *Matplotlib* para graficar señales uni-dimensionales (1D), con dos opciones de visualización: *plot* para tiempo continuo, *stem* para tiempo discreto. A diferencia de la librería de *Librosa,* es necesario definir un vector de tiempos previo a la visualización.

A continuación, se presenta el código en Python para las dos formas de visualización.

```
t = np.arange(0,len(audio)/fs,1/fs)
plt.rcParams["figure.figsize"] = (14,8)
ax = plt.subplot(2, 1, 1)
plt.plot(t[8000:8100], audio[8000:8100])
plt.title("Gráfica señal de voz utilizando plt.plot")
ax = plt.subplot(2, 1, 2)
plt.stem(audio[8000:8100])
plt.title("Gráfica señal de voz utilizando plt.stem")
```

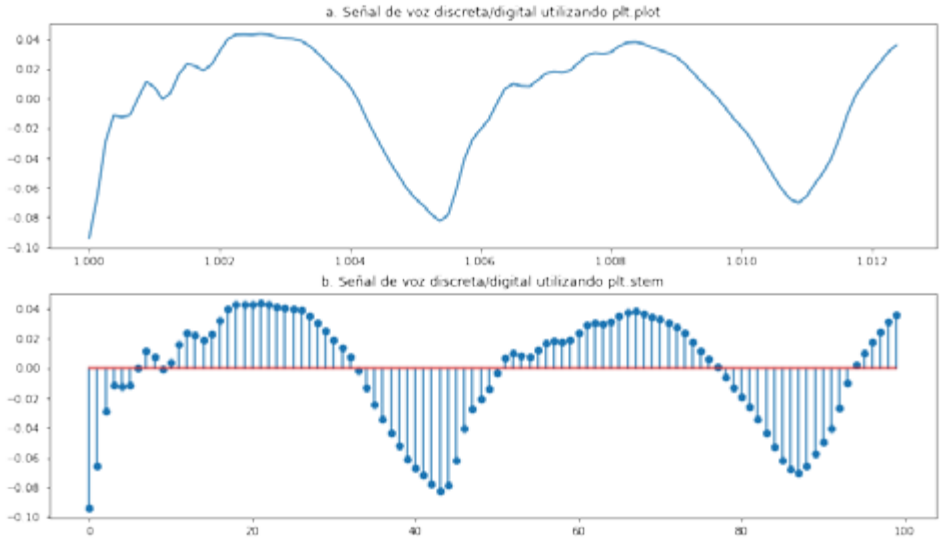

Figura 9. Ejemplo de señal de voz con dos formas distintas de visualización.

En la Figura 9a se graficaron 0.0125 segundos de la señal de voz de la Figura 4, comprendidos en el rango [1 1.0125) segundos, que corresponden a la interpolación de 100 muestras de la señal de voz en el rango [8000 8100). Aunque en este libro utilicemos en algunas ocasiones *plot* y en otras *stem*, el estudiante deberá siempre tener en cuenta que se están graficando señales discretas en el tiempo, con un número finito de bits de resolución.

1.2. CUANTIZACIÓN DE LAS MUESTRAS

Una vez se ha muestreado la señal, el siguiente paso (el cual se realiza casi de forma paralela en el conversor A/D) consiste *en representar mediante bits* a la amplitud de la señal discreta. Existen diversos formatos de representación de datos, por ejemplo, *magnitud, magnitud + signo, punto flotante,* entre otros. Supongamos que nuestro conversor trabaja con el formato *magnitud + signo,* donde el MSB *(Most Significant Bit: bit más significativo)* corresponde al signo del dato, y los restantes bits a la magnitud. De tal forma que, si el MSB es igual a 1, entonces la amplitud es negativa; en caso contrario, la amplitud es positiva.

Ahora bien, los conversores permiten trabajar con diferente número de bits de conversión por muestra, lo que se conoce como bits de resolución. A mayor cantidad de bits, la señal digital se escuchará más fiel a la señal análoga. Típicamente, podemos encontrar resoluciones de 8, 16, 24 y 32 bits.

Supongamos que nuestro audio que inicialmente se encontraba en el rango [-1 1] lo cuantizamos con 4 bits en formato *magnitud + signo*. Entonces, tenemos 3 bits

para la magnitud de la señal y 1 bit para el signo, de tal forma que cada incremento de amplitud de $1/2^3$ tendrá un nuevo código digital. Es decir, a todas las amplitudes del audio en el rango $[0 \quad 1/2^3)$ se les asigna el código 0000, a todas las amplitudes en el rango $[1/2^3 \quad 2/2^3)$ se les asigna el código 0001, y así sucesivamente. La Tabla 1 presenta la asignación de códigos por rangos de amplitud de la señal.

Tabla 1. Ejemplo de cuantización con 4 bits con formato magnitud + signo, para una señal en el rango [-1 1].

rango	código	Rango	código
$[-1 \quad -0.875)$	1111	$[0 \quad 0.125)$	0000
$[-0.875 \quad -0.75)$	1110	$[0.125 \quad 0.25)$	0001
$[-0.75 \quad -0.625)$	1101	$[0.25 \quad 0.375)$	0010
$[-0.625 \quad -0.5)$	1100	$[0.375 \quad 0.5)$	0011
$[-0.5 \quad -0.375)$	1011	$[0.5 \quad 0.625)$	0100
$[-0.375 \quad -0.25)$	1010	$[0.625 \quad 0.75)$	0101
$[-0.25 \quad -0.125)$	1001	$[0.75 \quad 0.875)$	0110
$[-0.125 \quad 0)$	1000	$[0.875 \quad 1)$	0111

De acuerdo con la Tabla 1, dos amplitudes que solo se diferencien en el signo (ej. -0.6 y 0.6) tendrán el mismo código excepto en su MSB (en este caso, 1100 y 0100). A medida que aumenta el número de bits de resolución, el rango de amplitudes que comparte el mismo código se va haciendo más pequeño. Por ejemplo, si la señal con rango análogo de [-1 1] la cuantizamos a 16 bits (15 de magnitud y 1 de signo), cada $1/2^{15}$ (es decir $30.5*10^{-6}$) tendrá un código digital distinto.

Para ilustrar el impacto de la cantidad de bits de resolución, utilicemos las mismas señales discretas de la sección anterior, la señal de voz y la de audio, para ilustrar el impacto de los bits de resolución en la calidad de la señal digital/discreta. El archivo audio.wav tiene 16 bits de resolución. Vamos a re-cuantizarlo a 8 bits (Figura 10), con el siguiente código en Python:

```
bits = 8
audio_8bit = (audio* 2**bits).astype(int)
audio_8bit = audio_8bit / 2**bits
librosa.display.waveplot(audio_8bit, sr=fs)
```

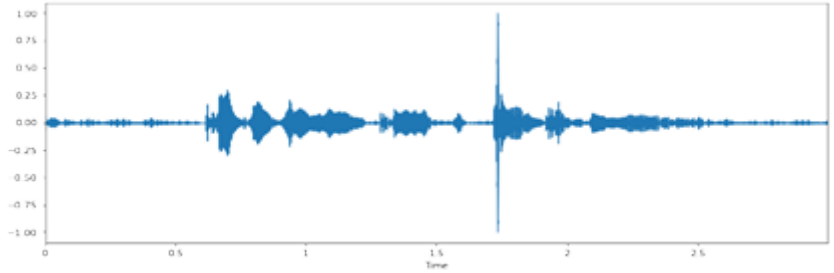

Figura 10. Ejemplo de señal de voz cuantizada a 8-bits.

y reproducimos la señal, así:

```
IPython.display.Audio(audio_8bit, rate=fs)
```

El efecto es que escuchamos *ruido de fondo* en la señal, pero el mensaje seguirá siendo legible. Ahora, disminuiremos la resolución a 6 bits (Figura 11), y compararemos los resultados con los obtenidos previamente.

```
bits = 6
audio_6bit = (audio* 2**bits).astype(int)
audio_6bit = audio_6bit / 2**bits
librosa.display.waveplot(audio_6bit, sr=fs);
IPython.display.Audio(audio_6bit, rate=fs)
```

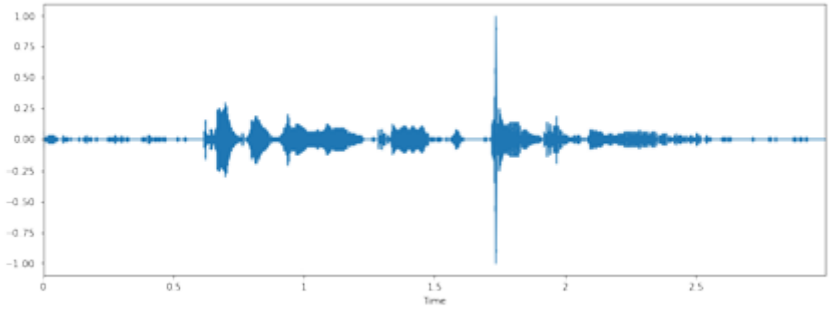

Figura 11. Ejemplo de señal de voz cuantizada a 6-bits.

En el audio re-cuantizado a 6 bits se escuchan *saltos de amplitud* en el mensaje. La calidad del audio en términos de legibilidad ha disminuido.

Finalmente, se re-cuantiza el audio a 3 bits (Figura 12). A diferencia de los casos anteriores, la señal re-cuantizada no tiene contenido inteligible (es decir, no se entiende lo que se dice), dado que la amplitud dista significativamente de la señal original cuantizada a 16 bits (Figura 4).

```
bits = 3
audio_3bit = (audio* 2**bits).astype(int)
audio_3bit = audio_3bit / 2**bits
librosa.display.waveplot(audio_3bit, sr=fs)
IPython.display.Audio(audio_3bit, rate=fs)
```

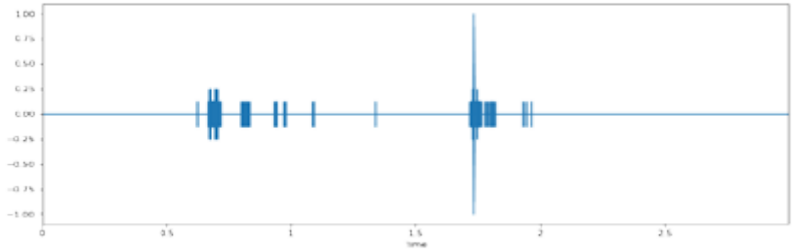

Figura 12. Ejemplo de señal de voz cuantizada a 3-bits.

Posteriormente, seleccionamos el archivo music.wav el cual tiene también 16 bits de resolución. Aplicaremos dos re-cuantizaciones: de 8 bits y de 3 bits.

```
# Re-cuantización a 8 bits del registro de música
bits = 8
music_8bit = (music* 2**bits).astype(int)
music_8bit = music_8bit / 2**bits
fig, ax = plt.subplots(nrows=2, sharex=True, sharey=True)
librosa.display.waveplot(music_8bit, sr=fs2, ax=ax[0])
ax[0].set(title='Música re-cuantizada a 8 bits')
ax[0].label_outer()

# Re-cuantización a 3 bits del registro de música
bits = 3
music_3bit = (music* 2**bits).astype(int)
music_3bit = music_3bit / 2**bits
librosa.display.waveplot(music_3bit, sr=fs2, ax=ax[1])
ax[1].set(title='Música re-cuantizada a 3 bits')
ax[1].label_outer()
```

Con la re-cuantización a 8 bits (Figura 13a), la señal es muy similar a la original cuantizada a 16 bits (Figura 6); mientas que la re-cuantizada a 3 bits (Figura 13b), tanto gráficamente como de forma auditiva, se aleja de la señal original.

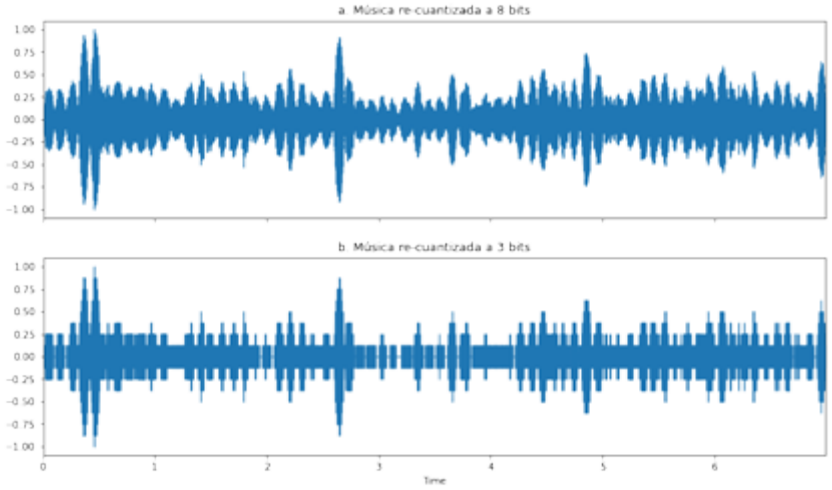

Figura 13. Ejemplo del efecto de re-cuantización de la señal de música a 8 bits y a 3 bits.

Por lo anterior, es evidente que la selección de la cantidad de bits de resolución juega un papel muy importante en la calidad de la señal discreta/digital. En la siguiente subsección abordaremos las implicaciones que tiene a nivel de costo de almacenamiento y de transmisión el valor de bits de resolución seleccionado.

1.3. COSTO DE ALMACENAMIENTO/TRANSMISIÓN EN TÉRMINOS DE LA FRECUENCIA DE MUESTREO Y NÚMERO DE BITS DE RESOLUCIÓN

Hasta aquí hemos evidenciado la importancia de seleccionar adecuadamente el valor de frecuencia de muestreo y de bits de resolución cuando vamos a convertir una señal continua/análoga en discreta/digital. Recordemos que la notación *"continua"* y *"discreta"* hace alusión a la variable independiente (típicamente el tiempo), mientras que *"análoga"* y *"digital"* corresponde con la amplitud de la señal (ej. voltios., amperes, entre otros). Si la cantidad de valores en un rango de tiempo es infinita, la señal es continua; en caso contrario, es discreta. De forma similar, si la cantidad de valores diferentes de amplitud en un rango es infinita, la señal es análoga; en caso contrario, es digital.

Aunque podríamos pensar que tanto la f_s seleccionada como los bits de resolución deberían ser los más altos posibles en beneficio de la calidad de la señal (similitud con la señal continua/análoga original), debemos tener presente que existe un costo asociado con el almacenamiento y la transmisión de la señal. Este concepto lo explicaremos a través de dos casos.

Caso 1:

Supongamos que se ha digitalizado una señal de voz de 2.777 horas (exactamente 10.000 segundos), con $f_s = 24$ kHz y 32 bits de resolución. Entonces, la cantidad de bits total de la señal digital/discreta es:

$$cantidad = Time * f_s * res \ [bits] \qquad \textbf{Ecuación 2}$$

Donde Time es la duración de la señal en segundos, f_s es la frecuencia de muestreo, y res es la cantidad de bits de resolución.

De tal forma que, $cantidad$ = 10.000 $*$ 24.000 $*$ 32 = 7.68 Gb, que corresponde a 960 MB.

Caso 2:

La misma señal de voz del Caso 1 se digitalizó con $f_s = 8$ kHz y 16 bits de resolución. Entonces, $cantidad$ = 10.000 $*$ 8.000 $*$16 = 1.28 Gb, que equivale a 160 MB.

Supongamos ahora que nuestro plan de Wi-Fi es de 10 MBps (donde Bps: bytes por segundo) y queremos descargar la señal de voz que se encuentra en dos páginas de internet. La primera página utilizó los parámetros de conversión del Caso 1; mientras que la segunda página utilizó los parámetros de conversión del Caso 2. Enton-

ces, la descarga del archivo en la primera página de internet tomaría 96 segundos (1 minuto y 36 segundos), mientras que, en la segunda página de internet tomaría 16 segundos. Es evidente que preferiríamos descargar el archivo de la segunda página de internet porque nos tomaría la sexta parte en relación con el tiempo de descarga en la primera página.

Pero ¿la calidad de la señal discreta/digital obtenida con los parámetros de conversión del Caso 2 es lo suficientemente buena? La respuesta es sí, dado que, tanto la f_s como la resolución son adecuados para señales de voz. No es necesario discretizar una señal que solo contiene voz con una $f_s = 24\ kHz$, dado que, como vimos previamente, la mayor parte de la energía de la señal se encuentra en las frecuencias inferiores a $1\ kHz$. Adicionalmente, la resolución de 16 bits permite cambios en el código digital para valores de amplitud muy pequeños.

Como conclusión, los valores de f_s y bits de resolución no deberían ser tan pequeños que nos degraden la calidad de la señal, pero tampoco excesivamente altos, que impliquen altos costos de almacenamiento y/o transmisión de la señal.

1.4. EFECTO EN EL ESPECTRO DE MUESTREAR UNA SEÑAL DE TIEMPO CONTINUO

En el canal de YouTube[1] podrás encontrar el video titulado *"Espectro señales discretizadas"* en el que se explica paso a paso el efecto de muestrear una señal continua en términos de su espectro. Este concepto lo explicaré de forma matemática a continuación.

Primero, partimos de una señal continua en el dominio del tiempo, la cual posee un número infinito de valores de tiempo en el rango de $[t_i \quad t_f]$, donde t_i es el tiempo inicial de la señal, y t_f es el tiempo final. Por ejemplo, supongamos que $t_i = 0\ s$, mientras que $t_f = 10\ s$. Esa señal la vamos a denominar $x(t)$ y su espectro $X(f)$.

Es decir,

$$x(t) \overset{FT}{\rightarrow} X(f) \qquad \textbf{Ecuación 3}$$

Supongamos que el espectro de la señal $x(t)$ está comprendido en el rango $[0 \quad 4]kHz$, por lo que decidimos muestrear la señal con f_s = 8 kHz. La forma de hacerlo es multiplicar $x(t)$ con un tren de impulsos periódico de amplitud igual a 1 y $T = 1/f_s$, que denominaremos $m(t)$. En nuestro caso, el periodo del tren de impulsos es $T=1/8kHz=125\ \mu s$. El espectro de m(t) lo denominaremos $M(f)$, el cual corresponde a otro tren de impulsos cuya amplitud es $1/T$ y espaciado cada f_s.

1 https://www.youtube.com/channel/UCrasAFtm_6B9vOIShGtI1ig

Es decir,

$$m(t) \overset{FT}{\rightarrow} FT\ M(f) \qquad \text{Ecuación 4}$$

El efecto en el dominio del tiempo de multiplicar $x(t)$ con $m(t)$, es que la señal continua queda muestreada cada T segundos, obteniendo una señal discreta que denominaremos $x[n]$. En el dominio de la frecuencia, el efecto es la convolución entre los espectros de $X(f)$ y $M(f)$, es decir, se generan "réplicas" del espectro $X(f)$ cada f_s Hz. A este espectro resultante lo denominaremos $X_m(f)$, así:

$$x[n] \overset{DFT}{\rightarrow} X_m(f) \qquad \text{Ecuación 5}$$

El efecto de réplicas en el espectro se explica recordando que cuando se convoluciona una señal por un impulso desplazado en k, el resultado es la misma señal desplazada en k. De tal forma que la convolución de $X(f)$ con el impulso ubicado en el origen es el mismo espectro $X(f)$; la convolución de $X(f)$ con el impulso ubicado en f_s es $X(f - f_s)$; la convolución de $X(f)$ con el impulso ubicado en $2\ f_s$ es $X(f - 2\ f_s)$; y así sucesivamente. Teniendo en cuenta que la señal $m(t)$ contiene infinitos impulsos separados f_s, entonces, la cantidad de réplicas de $X(f)$ es también infinita y están separadas f_s. Adicionalmente, su amplitud se verá afectada por el valor $1/T$.

Hasta aquí, vamos a resumir lo explicado anteriormente:

Tabla 2. Muestreo con tren de impulsos de duración infinita y su efecto en frecuencia.

Dominio del Tiempo (continuo o discreto)	Dominio de la Frecuencia
$x(t)$	$X(f)$
$m(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT), k \in \mathbb{Z}$	$M(f) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(t - kf_s), k \in \mathbb{Z}$
$\mathbb{Z}$ representa el conjunto de números enteros	$\mathbb{Z}$ representa el conjunto de números enteros
$x[n] = x(t) * m(t)$	$X_m(f) = X(f) \otimes M(f)$
$*$ significa multiplicación (efecto: señal muestreada cada T segundos)	$\otimes$ significa convolución (efecto: réplicas del espectro $X(f)$ cada f_s)

Ahora bien, teniendo en cuenta que en la práctica el tren de impulsos es de duración finita, podemos multiplicar $m(t)$ por una ventana $w[n]$, para limitar la duración del tren de impulsos en el rango $[t_i \quad t_f]$. Entonces, en el dominio de la frecuencia, el espectro del tren de impulsos, $M(f)$, se convoluciona con el espectro de la ventana, $W(f)$.

Dado que existen diversos tipos de ventana y que cada una tiene un espectro diferente, se expresará de forma general, tanto la señal en el dominio del tiempo, como en el dominio de la frecuencia, así:

$$w[n] \overset{DFT}{\rightarrow} W(f) \qquad \text{Ecuación 6}$$

En la Tabla 3 se presenta el efecto de muestreo de la señal $x(t)$ con el tren de impulsos de duración finita. Al final de todo este proceso, el espectro de la señal $x(t)$ no solamente se replica, sino que se distorsiona ligeramente, debido a la convolución en el dominio de la frecuencia entre $(X(f) \otimes M(f))$ con $W(f)$.

Antes de transmitir la señal muestreada, se aplica un filtro pasa-bajo, para obtener únicamente la réplica ubicada en el origen.

Tabla 3. Muestreo con tren de impulsos de duración finita y su efecto en frecuencia.

Dominio del Tiempo	Dominio de la Frecuencia
$x(t)$	$X(f)$
$m(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT), k \in \mathbb{Z}$	$M(f) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(t - kf_s), k \in \mathbb{Z}$
$w[n]$	$W(f)$
$x[n] = x(t) * m(t) * w[n]$	$X_m(f) = X(f) \otimes M(f) \otimes W(f)$
$*$ *significa multiplicación (efecto: señal muestreada cada T)*	$\otimes$ *significa convolución (efecto: réplicas del espectro $X(f)$ cada f_s con una ligera distorsión)*