TRANSMISIÓN DE DATOS 2007/08			
Práctica 2: Codificación PCM		**Grupo**	
		Puesto	
Apellidos, nombre		**Fecha**	
Apellidos, nombre		**29/31 de octubre y 8 de noviembre de 2007**	

El objetivo de esta práctica es familiarizar al alumno con la codificación de forma de onda (con pérdidas) mediante técnicas de Modulación por Impulsos Codificados –MIC- (Pulse Code Modulation – PCM).

Para llevar a cabo la práctica, desarrolle cada ejercicio en un fichero matlab dentro del directorio P2.

Se proporcionan dos códigos Matlab accesibles desde la página de la asignatura:

- `Grabando.m`: permite grabar un fichero de audio discreto (muestreado a 8 Khz) de 4 segundos de duración y almacenarlo en el fichero `sample_audio.mat`
- `Leyendo.m`: Es una función que permite leer un fichero de audio discreto (`sample_audio.mat`) , lo "sonifica" y dibuja sus gráficas características. Además devuelve un vector con la secuencia contenida en el fichero.

Todos estos ficheros están accesibles en el archivo correspondiente a la práctica 2 en la página web de la asignatura.

Justo antes de finalizar la práctica, comprima todo el directorio P2 en un fichero TxDatosP2GXxx.zip (siendo X el grupo –A, B o C- y xx el número de pareja), conéctese al sistema de entrega de prácticas de la Intranet y entréguelo en el grupo que corresponda (A, B o C). Guárdese adicionalmente una **copia personal**, para posibles futuras reutilización del código en prácticas posteriores.

2.1 Codificación PCM uniforme

La codificación PCM uniforme es el método más sencillo de codificación por forma de onda y consiste en la aplicación directa de los conceptos de cuantificación uniforme.

El algoritmo básico consiste en dividir el rango dinámico en tantos intervalos uniformes como el número de niveles proporcionados por el número de bits disponibles para codificar cada muestra, y asignar un intervalo a cada muestra. La reconstrucción se hace al valor de reconstrucción de cada intervalo (en este caso al nivel medio de cada intervalo).

2.1.1 Ejercicio 1: Codificador Uniforme simétrico de 8 bits

Implemente una función que calcule los índices (desde 0 hasta el número máximo) de intervalo a los correspondientes una colección de muestras que respete la siguiente sintaxis:

```
function [Scuan] = getIndicesUniformes (S,nNiveles,vinf,vsup)
```[1]

- **Recibe:**
  - S == Secuencia de audio grabada.
  - nNiveles == Número de niveles de cuantificación.
  - vinf == Valor mínimo de la señal.
  - vsup == Valor máximo de la señal.
- **Devuelve:**
  - Scuan == Índices de los intervalos de cuantificación calculados para la señal de entrada.

Haciendo uso de esta función escriba el código de una función que implemente un cuantificador uniforme simétrico, sin nivel de reconstrucción igual a 0, de n bits, que será guardado en `PCMUniforme.m`. La sintaxis de la función será la siguiente:

```
function [Scuan] = PCMUniforme (S,nbits,v)
```

- **Recibe:**
  - S == Secuencia de audio grabada.
  - nbits == Número de bits a utilizar para la codificación de cada muestra.
  - v == Valor de sobrecarga.
- **Devuelve:**
  - Scuan == Índices de los intervalos de cuantificación calculados para la señal de entrada.

Implementar una función `decodificaPCM` para llevar a cabo la reconstrucción de una señal cuantificada que respete la siguiente sintaxis:

```
function [Srecon] = decodificaPCM (Scuan,nbits,v)
```

- **Recibe:**
  - Scuan == Índices de los intervalos cuantificación calculados para la señal original.
  - nbits == Número de bits utilizados en la cuantificación de la señal original
  - v == Valor de sobrecarga utilizado en la cuantificación de la señal original.
- **Devuelve:**
  - Srecon == Señal reconstruida**.**

Recomendaciones:

Compruebe la corrección de cada una de las funciones con secuencias cortas de las que pueda calcular analíticamente el resultado esperado.

---

[1] Recuerde comprobar dentro de esta misma función que ninguno de los índices resultantes es mayor que `nNiveles-1` ni menor que 0.

Recuerde que la potencia de matlab reside en su capacidad para el calculo matricial (i.e. Puede dividir todos los elementos de una matriz o vector por un escalar con UNA sola operación).

Dibuje e indique los valores de decisión y reconstrucción de un cuantificador uniforme simétrico (sin nivel de reconstrucción igual a 0) de 2 bits, y valor de sobrecarga V.

Dibuje e indique los valores de decisión y reconstrucción de un cuantificador uniforme simétrico de 3 bits, y valor de sobrecarga V.

Calcule los códigos, valores de reconstrucción y error de las siguientes muestras:

- V1(2)=0.7507 voltios con valor de sobrecarga 2 voltios (cuantificador 2 bits)

- V1(3)=0.7507 voltios con valor de sobrecarga 2 voltios (cuantificador 3 bits)

- V1(8)=0.7507 voltios con valor de sobrecarga 2 voltios (cuantificador 8 bits)

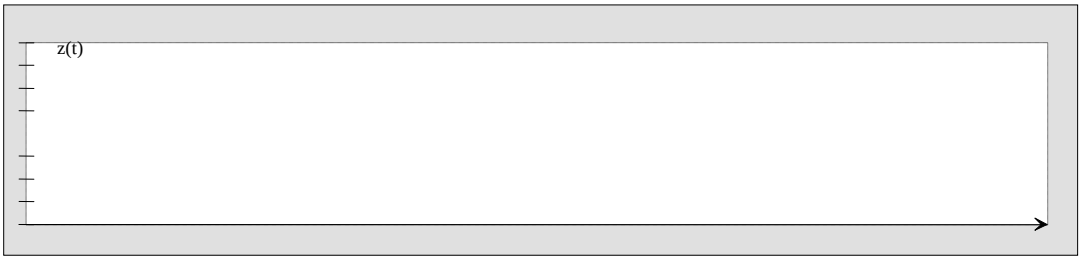
- V2(2)=0.2 voltios con valor de sobrecarga 1 voltio (cuantificador 2 bits)

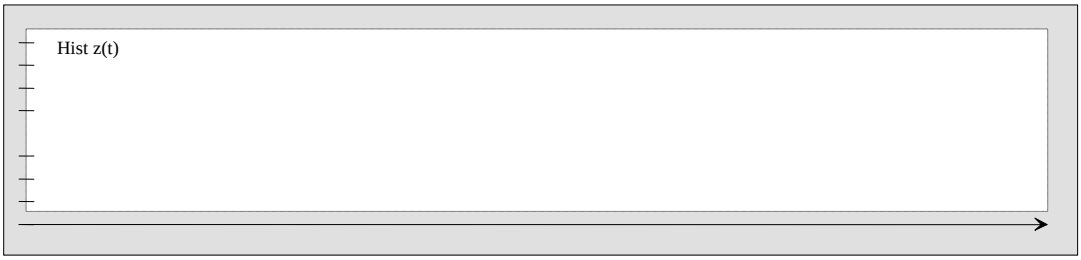
- V1(3)=0.2 voltios con valor de sobrecarga 1 voltio (cuantificador 3 bits)

- V1(8)=0.2 voltios con valor de sobrecarga 1 voltio (cuantificador 8 bits)

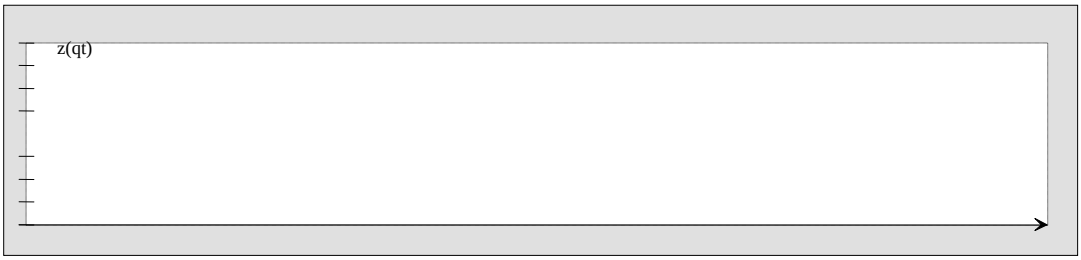
A partir de las funciones implementadas anteriormente, desarrolle un script Matlab con nombre ***'Ejercicio1.m'*** en el que se lea el fichero de audio `sample_audio.mat` y se lleve a cabo su cuantificación y reconstrucción (con 8 bits y valor de sobrecarga adecuado para evitar el error en el proceso de cuantificación por saturación) y dibuje las gráficas correspondientes a las señales cuantificada y reconstruida, indicando los valores de los ejes. Dibuje las gráficas obtenidas:

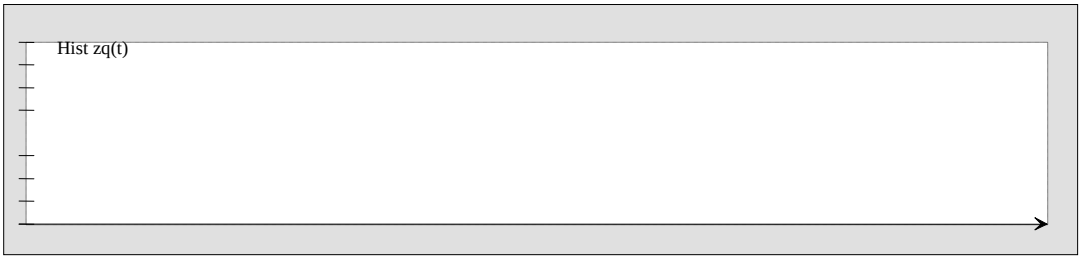
- Señal original



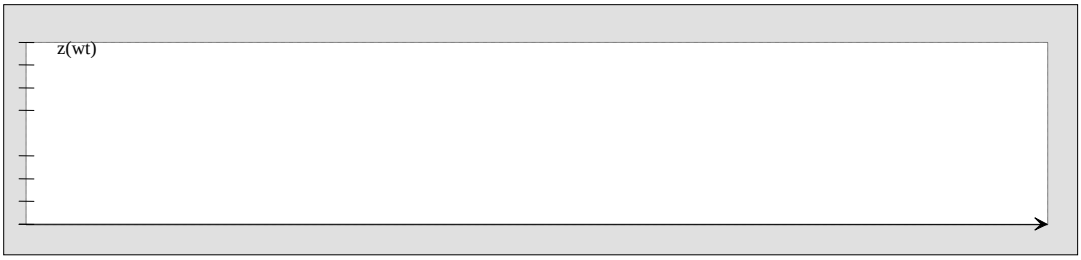


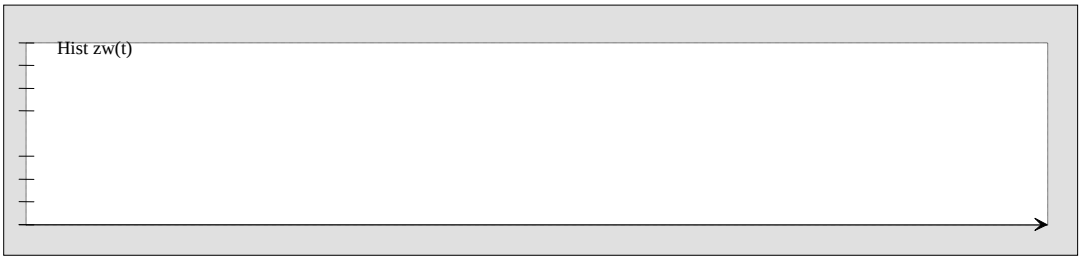
- “Señal” cuantificada



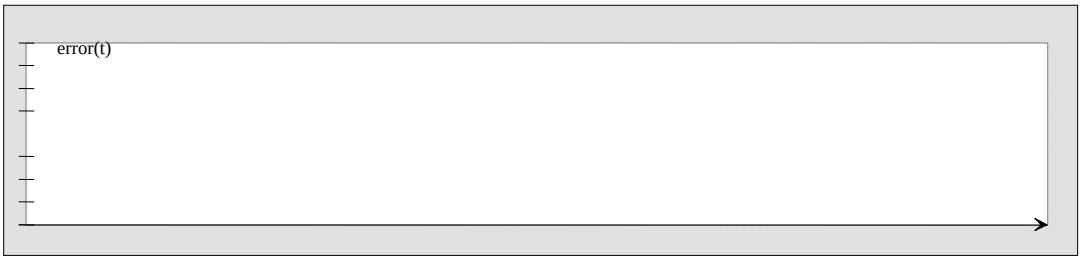


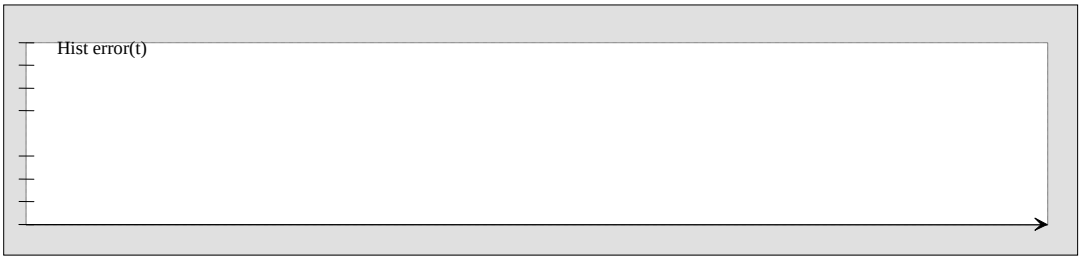
- Señal reconstruida





- Señal de error





Indique la potencia de error (Perror = sum(error.^2))

Comente los resultados

Codificación PCM no uniforme

La codificación PCM no uniforme se basa en dividir el rango en intervalos de anchura variable, de forma que con el mismo número de bits (niveles) se pueda dar más precisión en las zonas donde hay una mayor probabilidad (densidad) de valores, lo que hace disminuir la potencia total de ruido al disminuir el valor de la contribución de las valores más probables. Generalmente se implementan mediante técnicas de "compansión" (compresión-expansión).

Las técnicas más usadas son los cuantificadores no uniformes robustos los cuales, en lugar de minimizar la distorsión, buscan lograr una relación señal-a-ruido(ruido de cuantificación) constante. Estos cuantificadores tienen funciones de compresión logarítmicas. Adicionalmente se implementan como aproximaciones a las funciones (conocidas como *leyes*) teóricas. La norma utilizada en telefonía digital fija en Europa sigue la ley-A, aproximada mediante una función rectilínea que se especifica en la Recomendación G.711 de la UIT-UTI.

La Recomendación G.711, divide el rango (simétrico respecto al origen) en 13 segmentos rectilíneos. Si nos centramos en la zona positiva, se generan 7 segmentos (realmente 6+1+1). El segmento 7 ocupa la mitad del rango normalizado a 1 (Unidad de Tensión Normalizada –UTN-), esto es, de 1 a 0.5. El segmento 6 de 0.5 a 0.25, y así hasta llegar al segmento 1. El segmento 1 se divide en dos segmentos de igual tamaño (el 1 y el 0) por lo que suele hablar de 6+1+1 segmentos. Dentro de cada segmento no uniforme, se cuantifica el rango mediante 16 intervalos de cuantificación uniformes.

La codificación consiste en asignar un bit al signo (1 si positivo), 3 bits al segmento, y 4 bits al intervalo de cuantificación.

2.1.2 Ejercicio 2: Codificador G.711

Para el desarrollo del codificador G711 se implementarán por separado las funciones para calcular los valores para el signo, intervalo y segmento. Lleve a cabo la implementación de dichas funciones como se indica a continuación:

```
function [signos] = getSigno (Snorm)
```

- **Recibe:**
  - `Snorm` == Secuencia de audio grabada NORMALIZADA.
- **Devuelve:**
  - `signos` == Signos de las muestras de entrada: Mayor o igual que 0→ '1' , Menor que 0 → '0'.

Escriba la función *getSegmento* que devuelve los índices de los segmentos a los que pertenece una colección de muestras normalizadas (para ello deberá definir a mano dentro de la función los límites para cada segmento)

```
function [segmentos] = getSegmento (Snorm)
```

- **Recibe:**
  - `Snorm` == Secuencia de audio grabada NORMALIZADA.
- **Devuelve:**
  - `segmentos` == Segmentos de las muestras de entrada: de 0 a 7.

A partir de la información de a qué segmento pertenece cada muestra y haciendo uso de la función implementada anteriormente `getIndicesUniformes` implemente la función `getIntervalo` que devolverá el resultado de realizar una cuantificación uniforme con 16 niveles (4 bits) de cada muestra sobre el segmento que le corresponda:

```
function [intervalos] = getIntervalo (Snorm,segmentos)
```
[2]

- **Recibe:**
  - `Snorm` == Secuencia de audio grabada NORMALIZADA.
  - `segmentos` == Segmentos de las muestras de entrada: de 0 a 7.
- **Devuelve:**
  - `intervalos` == Intervalos de las muestras de entrada: de 0 a 15.

Implemente una función que calcule los valores de cuantificación correspondientes a una colección de muestras mediante el uso de las tres funciones anteriores y que respete la siguiente sintaxis:

```
function [Scuan] = getCuantificacionG711 (Snorm)
```

- **Recibe:**
  - `Snorm` == Secuencia de audio grabada NORMALIZADA.
- **Devuelve:**
  - Scuan == Valores de cuantificación para la señal de entrada[3]

---

[2] Utilice getIndicesUniformes para el cálculo de los índices del intervalo.

Escriba el cógigo de una función `PCMG711` en la que se lleve a cabo el proceso de normalización y cuantificación de una señal mediante las funciones implementadas anteriormente:

```
function [Scuan] = PCMG711 (S,v)
```

- **Recibe:**
  - S == Secuencia de audio grabada.
  - v == Valor de sobrecarga.
- **Devuelve:**
  - Scuan == Valores de cuantificación para la señal de entrada.

Haciendo uso de los conceptos aplicados para el proceso de codificación implemente una función `decodificaG711` que lleve a cabo la decodificación de una señal cuantificada previamente. En caso de considerarlo necesario implemente las funciones auxiliares que considere oportuno. La sintaxis de la función será la siguiente:

```
function [Srecon] = decodificaG711 (Scuan,v)
```

- **Recibe:**
  - Scuan == Índices de los intervalos de entrada para la señal de entrada.
  - v == Valor de sobrecarga.
- **Devuelve:**
  - Srecon == Señal reconstruida.

Recomendaciones:

Elija un par de valores de muestras e inicie el desarrollo probando sobre una señal ficticia de una muestra (cuyo resultado haya calculado teóricamente).

La secuencia de entrada `S` *se obtendrá mediante la función* `Leyendo.m`

La función `getCuantificacionG711` *espera una señal de entrada que varíe entre -1 y 1.*

Calcule analíticamente los códigos, valores de reconstrucción y error de las siguientes muestras:

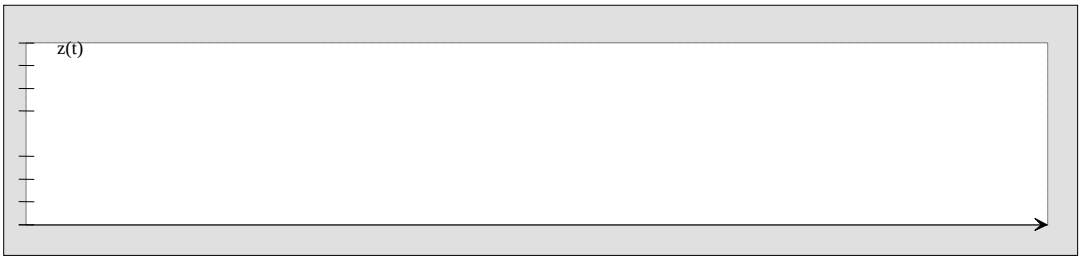
- V1(G)=0.7507 voltios con valor de sobrecarga 2 voltios

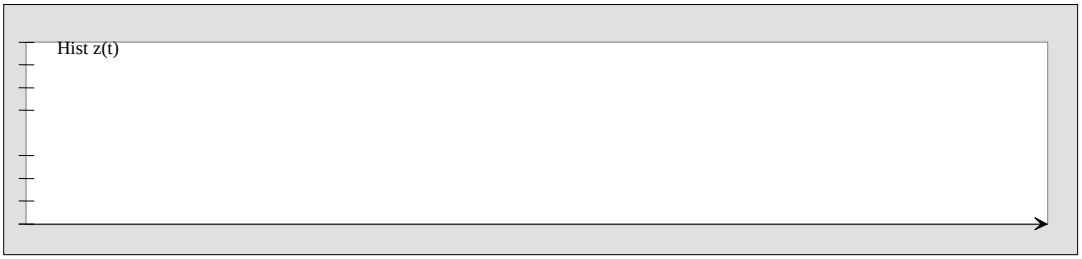
[3] secuencia de valores 0-255 [signo(1bit) segmento(3bits) intervalo(4bits)]

- V1(G)=0.2 voltios con valor de sobrecarga 1 voltio (cuantificador 8 bits)

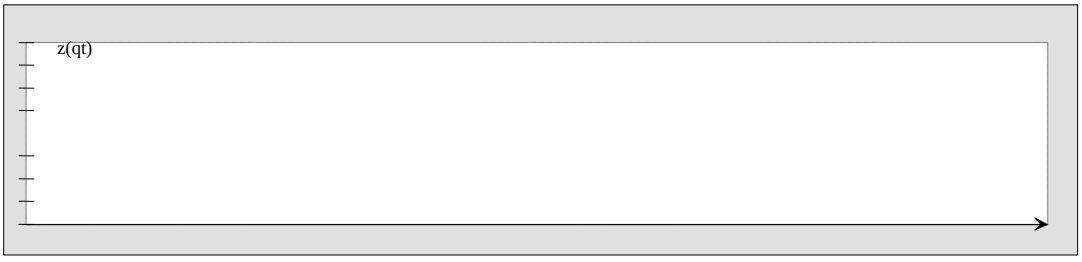
Desarrolle un script Matlab con nombre ***'Ejercicio2.m'*** en el que se lea el fichero de audio `sample_audio.mat` y se lleve a cabo su cuantificación y reconstrucción mediante las funciones implementadas y dibuje las gráficas correspondientes a las señales cuantificada y reconstruida, indicando los valores de los ejes. Dibuje las gráficas obtenidas:

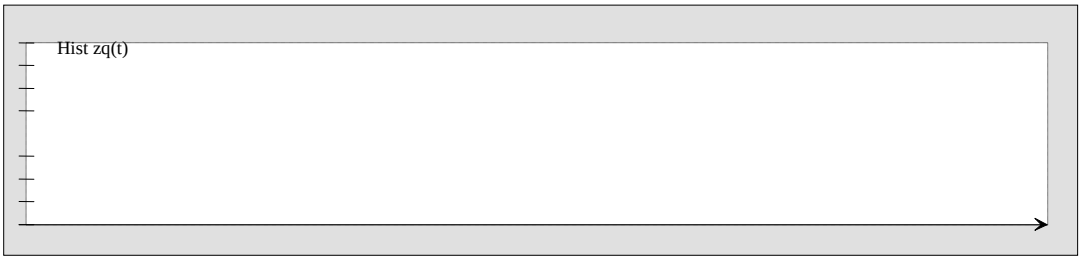
- Señal original



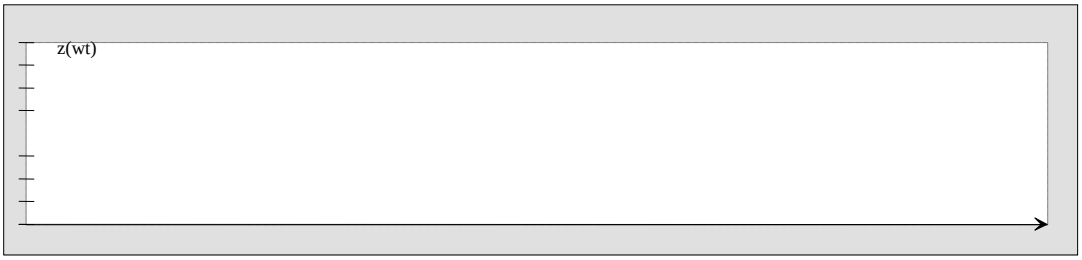


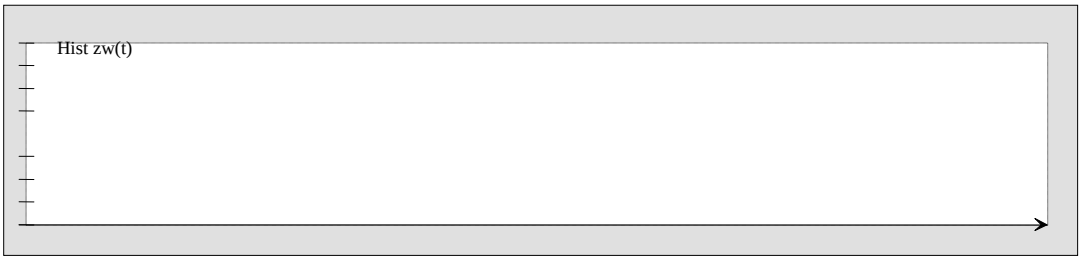
- "Señal" cuantificada



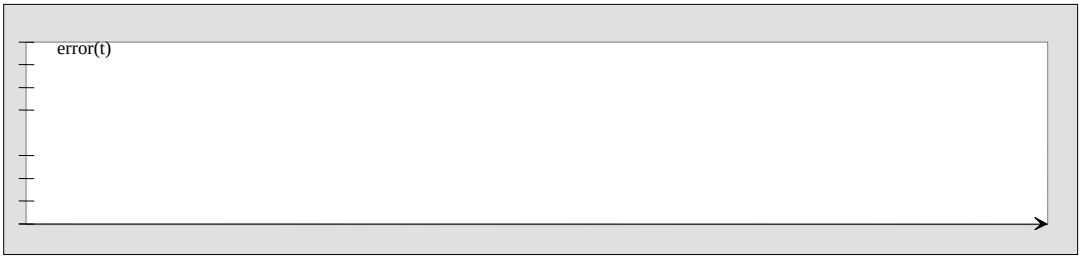


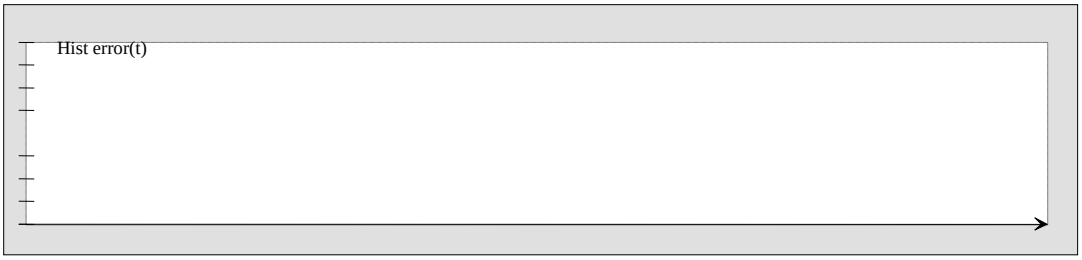
- Señal reconstruida





- Señal de error





Indique la potencia de error

Comente los resultados y compárelos con los del cuantificador uniforme

2.2 Transmisión de la Codificación

En el fichero de la práctica se proporciona la función `getSecuenciaBits` con la siguiente especificación:

```
function [secuencia] = getSecuenciaBits (Scuan,nbits)
```

- **Recibe:**
  - Scuan == Señal cuantificada.
  - nbits == Número de bits por muestra en la cuantificación.
- **Devuelve:**
  - secuencia == Vector fila en el que cada posición se corresponde con un bit de la secuencia Scuan en formato binario.

Desarrolle un script Matlab con nombre ***'Ejercicio3.m'*** en el que se lea el fichero de audio `sample_audio.mat` y se lleve a cabo la codificación uniforme con 4 y 8 bits y la G711 aplicando la función `getSecuenciaBits` a los resultados obtenidos.

Comente los resultados: