

Práctica 3. Cuantificación y codificación PCM

Resumen

La cuantificación es la base de la codificación con pérdidas. En esta práctica estudiaremos la eficiencia de la cuantificación uniforme de señales de voz. Posteriormente, implementaremos un codificador y decodificador PCM de voz que utiliza cuantificación uniforme y códigos de longitud variable.

1. Cuantificación uniforme de voz

El fichero `vt1.wav` contiene la secuencia de bits generada por un convertidor A/D al codificar una señal de voz de banda estrecha. El convertidor A/D está formado por un convertidor C/D con frecuencia de muestreo de 8 kHz, un cuantificador uniforme de 2^{16} intervalos y un codificador sin pérdidas que utiliza un código de longitud fija de 16 bits. La tasa binaria (expresada en bps) de la voz codificada en `vt1.wav` es

$$R_0 = f_s \times L = 8 \cdot 10^3 \times 16 = 128 \cdot 10^3 \text{ bps.}$$

🔧 Decodifica la secuencia de voz de `vt1.wav` utilizando `audioread` (sin la opción `native`) y escúchala utilizando `soundsc`.

Si consideramos que 128 kbps es una tasa binaria demasiado elevada, podemos reducirla utilizando *compresión* (figura 1). En esta práctica, consideraremos la compresión de voz con PCM, esto es, la codificación de la secuencia $x[n]$ proporcionada por el convertidor A/D utilizando un cuantificador y un codificador sin pérdidas. En concreto, el cuantificador utilizado será uniforme y el codificador sin pérdidas será un código de longitud fija.

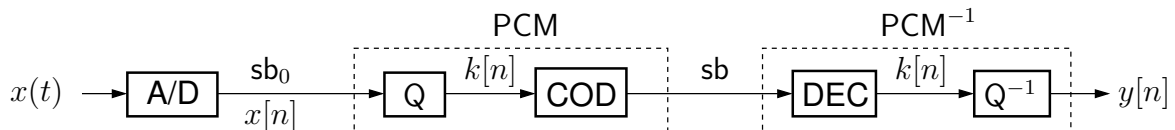



Figura 1. Compresión de $x[n]$ utilizando PCM.

Si Δ es el paso de cuantificación y x la amplitud de una muestra a cuantificar, podemos cuantificar x con

$$k = \left\lfloor \frac{x}{\Delta} \right\rfloor \quad (1)$$

donde k el índice de cuantificación resultante ($k = Q(x)$). Para implementar el cuantificador Q del codificador utilizaremos la función `floor` de MATLAB. Dado que MATLAB permite que el argumento de `floor` sea un array, podemos codificar todas las muestras de una secuencia o una matriz con una única instrucción `floor`.

 Dibuja la función $k = Q(x)$.

 ¿Cuál es la operación que realiza el decuantizador sobre k para obtener y ?

Observa que en el cuantizador definido en (1) no hay saturación: el valor de k crece indefinidamente con el valor de x . Como $x[n]$ proviene de un convertidor A/D, solo puede tomar valores de amplitud en un intervalo finito. Así, cuando leemos una secuencia $x[n]$ de un fichero de audio WAVE con `audioread` (sin la opción `native`), tenemos que

$$-1 \leq x[n] < 1.$$

Nuestro codificador PCM codifica los índices de cuantificación utilizando un código de longitud fija. Si R es la tasa binaria del código (en bits/muestra), podemos codificar $M = 2^R$ índices de cuantificación. Como la señal toma valores en $[-1, 1)$, repartiremos los M intervalos de cuantificación disponibles en $[-1, 1)$; por tanto, el paso de cuantificación será


$$\Delta = \frac{2}{M} = \frac{2}{2^R} = 2^{(1-R)}.$$

Cuanto mayor es R , menor es el valor de Δ y, consecuentemente, menor es la distorsión introducida. Al cuantizar $x[n]$ usando


$$k = \left\lfloor \frac{x}{2^{(1-R)}} \right\rfloor$$


se generan índices en el conjunto

$$\mathcal{K} = \{-2^{R-1}, -2^{R-1} + 1, \dots, 0, \dots, 2^{R-1} - 1\}.$$

 Vamos a comprobar todo lo anterior, escribiendo un script de MATLAB que realice las siguientes operaciones:

1. Obtén el valor de Δ para $R = 13$ bpm.
2. Obtén la secuencia del fichero `vt1.wav` utilizando `audioread` ($\rightarrow x[n]$).
3. Cuantiza la secuencia $x[n]$ ($\rightarrow k[n]$).
4. Decuantiza la secuencia de índices $k[n]$ ($\rightarrow y[n]$).
5. Muestra por pantalla la distorsión (o error de reconstrucción cuadrático medio).
6. Reproduce la secuencia decodificada $y[n]$ utilizando `sound` (use el valor de f_s).

 Reproduce $x[n]$ e $y[n]$. ¿Aprecias alguna diferencia?

 Ejecuta el script con R igual a 12, 10, 8, ... fijándote en la calidad de la voz decodificada en cada caso. ¿A partir de qué tasa binaria empiezas a percibir distorsión?

2. Codificación PCM de voz

En este apartado implementaremos un codificador PCM para secuencias de voz de banda estrecha y su correspondiente decodificador. Supondremos que la señal de voz a codificar $x[n]$ cumple $-1 \leq x[n] < 1$.

El codificador PCM a implementar utiliza cuantificación uniforme y un código de longitud fija. El diagrama de bloques del codificador-decodificador se muestra en la figura 2. El codificador PCM está formado por un cuantizador (que transforma cada muestra $x[n]$ en su correspondiente índice $k[n]$) y un código de longitud fija (que transforma cada índice $k[n]$ en una palabra de R bits). El decodificador PCM está formado por el decodificador sin pérdidas y el decuantizador.

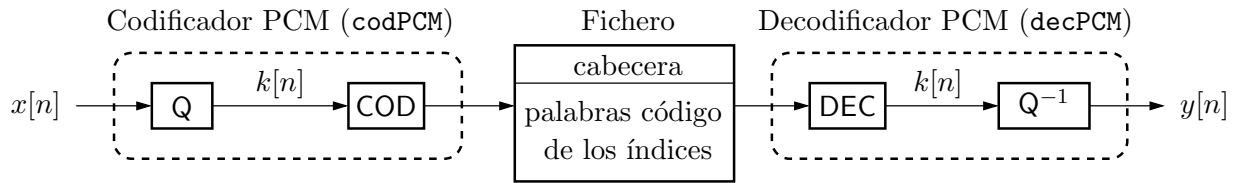


Figura 2. Codificador y decodificador PCM.

La secuencia de bits generada por el codificador PCM se almacena en un fichero. Este fichero contiene primero una cabecera con información relevante y, a continuación, las palabras código de los índices de cuantificación.

Escribe una función de MATLAB que implemente el codificador. La primera línea de esta función debe ser

```
function codPCM(x,R,fi)
```

donde:

- \mathbf{x} (*double*) es el vector con las muestras de la secuencia $x[n]$ a codificar;
- \mathbf{R} (*int*) es el número de bits con el que se codifica cada muestra de $x[n]$ ($1 \leq R \leq 15$);
- \mathbf{fi} es el nombre del fichero que contiene la secuencia de bits generada.

La función `codPCM` debe generar un fichero que, aparte de la secuencia de bits codificada, contenga al principio una cabecera con la siguiente información:

- El valor de \mathbf{R} codificado con 4 bits;
- El número de muestras de \mathbf{x} codificado con 32 bits.

El cuantizador del codificador, $Q(x)$, es el mismo que el de la sección 1, esto es,

$$k = \left\lfloor \frac{x}{2^{1-R}} \right\rfloor.$$

Los índices k generados con la expresión anterior son enteros entre -2^{R-1} y $2^{R-1} - 1$. Para facilitar la codificación sin pérdidas de dichos índices, primero los convertiremos en enteros sin signo sumando 2^{R-1} a cada índice:

$$k' = k + 2^{R-1}$$

Como tras la suma, los nuevos índices k' son enteros entre 0 y $2^R - 1$, dichos índices pueden codificarse en binario directo con R bits/índice.

Por ejemplo si $x = -0.1$ y $R = 6$, el cuantizador del codificador PCM obtiene

$$k = \lfloor -0.1/2^{-5} \rfloor = \lfloor -3.2 \rfloor = -4$$

y tras sumar $2^5 = 32$ a k obtenemos $k' = 28$ cuya codificación con 6 bits proporciona la palabra código 011100.

En el decodificador PCM, el decodificador sin pérdidas proporcionará los índices k' . Para obtener los índices k deberemos restar a cada uno de ellos 2^{R-1} :

$$k = k' - 2^{R-1}.$$

Así, la decodificación de la palabra código 011100 proporciona el entero $k' = 28$, y tras restar de $2^5 = 32$ a k' obtenemos el índice $k = -4$.

En el codificador PCM, la codificación de todos los índices k' y su escritura en el fichero puede realizarse con una única llamada a la función `fwrite` con el argumento `precision` especificado de manera adecuada.


Cuando `codPCM` escribe un entero sin signo en el fichero, el número de bits escritos debe depender del rango de valores de dicho entero. Así por ejemplo, escribir el valor de `R` (entero entre 1 y 15) usando `'uint8'` como el argumento `precision` de `fwrite` es ineficiente puesto que se escribirían 8 bits cuando 4 bits son suficientes para guardar el valor de `R`.

Solucionaremos este problema utilizando `ubitn` como el argumento `precision` de `fwrite`. En MATLAB, el tipo de datos `ubitn` representa enteros sin signo de un número *arbitrario* de bits n (n es un entero entre 1 y 64). El valor de n debe fijarse de acuerdo con el valor máximo que puede tomar la variable a escribir (el mayor entero sin signo de n bits es $2^n - 1$). Así, en el ejemplo anterior, escribiríamos `R` con la instrucción `fwrite(fid,R,'ubit4')`.

Si el número de bits a escribir está en una variable `m`, podemos generar la cadena `p = ['ubit' num2str(m)]` y usar `p` como el argumento `precisión` de `fwrite`.

Utilizando `codPCM` codifique `vt1` a 96 kbps.


 ¿Qué tamaño debería tener el fichero generado? bytes


 El tamaño del fichero generado en disco, ¿coincide con el teórico calculado?

Escribe una función de MATLAB que decodifique y reproduzca los ficheros generados por `codPCM`. La declaración de esta función debe ser

```
function y = decPCM(fi)
```

donde `fi` es el nombre del fichero con la secuencia de bits a decodificar e `y` (*double*) es la secuencia de voz decodificada.

 Utilizando `decPCM` decodifica el fichero y comprueba que la distorsión es casi inapreciable.

 Codifica (con `codPCM`) y decodifica (con `decPCM`) la secuencia de `vt1.wav` a varias tasas binarias. La variación de calidad en función de la tasa binaria debería ser la misma que apreció usando el script de la sección 1.