Práctica 4:

Codificación de señales de audio en subbandas

Sistemas de codificación y almacenamiento

Marina Rodríguez Rubio y Alejandro Pérez Martínez

12/11/2024

Índice

1. Ejercicio 1: Implementación de un codificador y descodificador PCM adaptable

La **Modulación por Impulsos Codificados Adaptable (APCM)** es una técnica de codificación digital que ajusta dinámicamente la cuantización de la señal según su amplitud local. Divide la señal en ventanas y calcula una ganancia para cada una, normalizando los valores antes de cuantificarlos con una resolución fija. Esto permite representar eficientemente señales con variaciones de amplitud, mejorando la relación señal-ruido (SNR) sin aumentar el número total de bits.

Listing 1: Codificador APCM

```
def encoderAPCM(data, wlen, dataRange, b, bg):
    code = []
    gainRange = [-1,1]

    for k in range(0,len(data),wlen):
        w = data[k : k + wlen] # ventana , w->window
        gain = max(abs(w)) # la ganancia es el mayoy de los valores absolutos
        w = w/gain #Aplica la ganancia a los datos actuales
        cw = encoderPCM(w,dataRange,b) # # Cuantifica (mapeo del codifcador) con b
             bits los datos yCodica con b bits los datos cuanticados
        cg = encoderPCM(gain,gainRange, bg) # Cuantica (mapeo del codicador) con
            bg bits la ganancia y Codica con bg bits la ganancia cuanticada
        code = np.concatenate((code, cg, cw)) # Acumula los datos de la ventana
            actual(al poner dos parentesis ni da error)

    return code
```

Listing 2: Decodificador APCM

```
def decoderAPCM(code, wlen, dataRange, b, bg):
    data = []
    gainRange = [-1,1]

    for k in range(0,len(code),bg + wlen * b):
        cg = code[k:k+bg] # Recupera la secuencia de bits correspondiente a la
            ganancia
        cw = code[k+bg:k+bg+wlen*b]
        gain = decoderPCM(cg,gainRange, bg) # Decodifica la ganancia y la
            cuantifica(mapeo del descodificador)
        w = decoderPCM(cw,dataRange,b) # Decodifica los datos y los cuantifica (
            mapeo del descodificador)
        w = w*gain # Aplica la ganancia a los datos actuales
        data = np.concatenate((data, w)) # Acumula los datos de la ventana actual

    return data
```

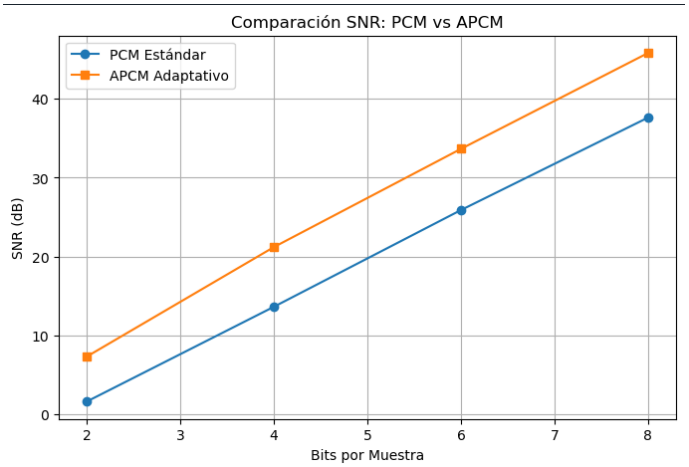


Figura 1: Comparación SNR.

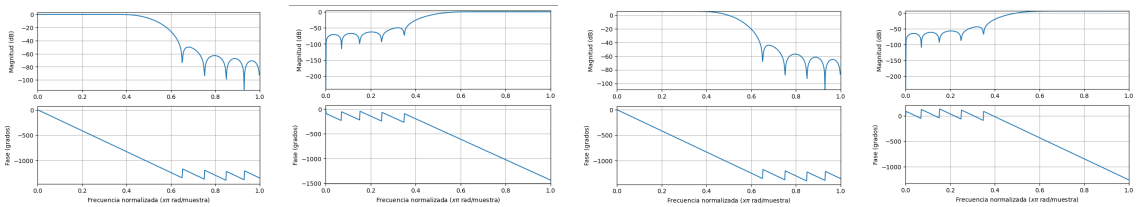
Aquí, se muestra una comparación entre los niveles de relación señal-ruido (SNR) obtenidos mediante codificación PCM estándar y APCM en función del número de bits por muestra. Se observa que en ambos métodos el SNR aumenta con el número de bits, como es de esperar, pero APCM ofrece un rendimiento significativamente superior para cada valor de bits por muestra, reflejando capacidad para adaptarse dinámicamente a las variaciones de amplitud de la señal. En contrapartida, APCM ocupa algo más de espacio dado que por cada ventana que condificamos tenemos que añadir la codificación de la ganancia, este junto a la mayor complejidad del algoritmo es el precio a pagar por esa mejora de la SNR.

2. Ejercicio 2: Cálculo de los coeficientes de la respuesta al impulso de los filtros QMF para la codificación en sub-bandas.

A continuación se muestra la función creada para el cálculo de los coeficientes:

Listing 3: Cálculo de la matriz DCT

```
def QMF_filters(length, fc):

    # FIR de paso bajo con ventana de Hamming
    h1 = signal.firwin(length, fc, window='hamming')

    # Generar el signo alternante (-1)^k vectorizado
    signs = (-1) ** np.arange(length)

    # Filtro de paso alto QMF
    h2 = h1 * signs

    # Ganancias k1,k2 segn ecuaciones
    k1 = 2 * h1
    k2 = -2 * h2

    return h1, h2, k1, k2
```



Filtro h1 Filtro h2 Filtro k1 Filtro k2

Figura 2: Módulo y fase de los 4 filtros usados en la codificación en subbandas.

Puede observarse que el filtro h_1 actúa como un paso baja, y que su correspondiente filtro de síntesis (k_1) presenta una ganancia en magnitud de aproximadamente 3 dB, de modo que la frecuencia de corte se sitúe en torno a 0 dB tras la reconstrucción. Esta misma relación ocurre entre los filtros h_2 y k_2, pero en este caso con un comportamiento pasa alta. Los 4 filtros presentan una atenuación considerable en la banda de rechazo.
Por otro lado destacar que, aunque la fase no es completamente lineal para ninguno de los 4 filtros, en la banda de paso si que lo es. De tal modo, lo que nos interesa es que obtengamos una fase lineal únicamente en la banda de paso, siendo la banda de rechazo de muy poca importancia dado que la atenuación en ella es muy grande.

3. Ejercicio 3:Análisis de la Respuesta en Frecuencia del Sistema de Subbandas y Estudio del Impacto del Filtro Paso-Baja h_1

A continuación vamos a ver la respuesta del sistema completo formado por los filtros del ejericicio 2,el sistema atiende al siguiente diagrama de bloques:

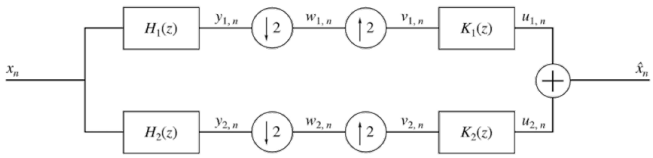


Figura 3: Esquema del sistema de codificación en subbandas con 2 canales.

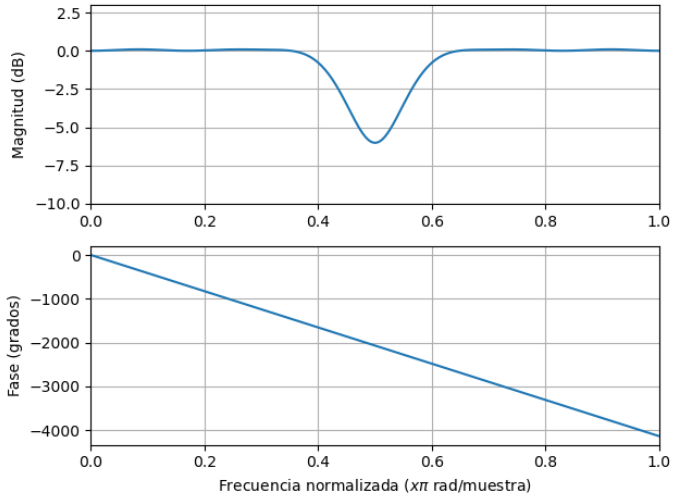
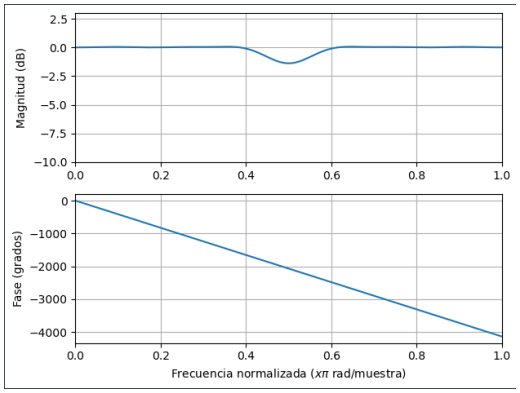
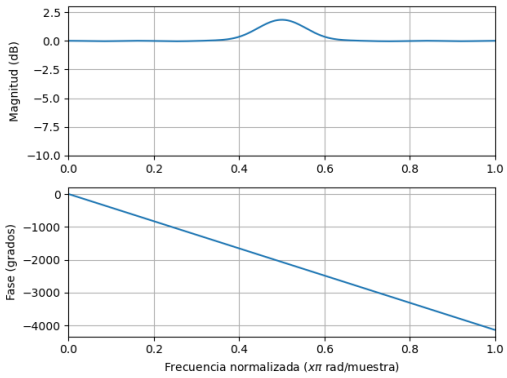


Figura 4: Respuesta del sistema completo

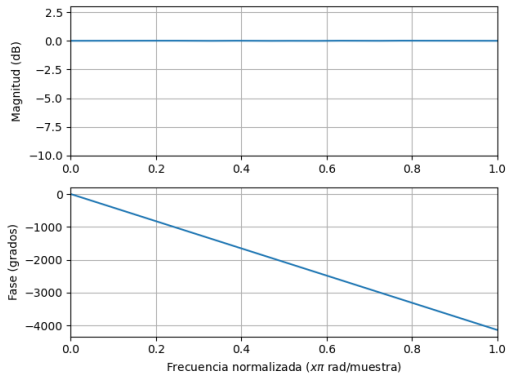


$f_c = fc + 5\,\% f_c$

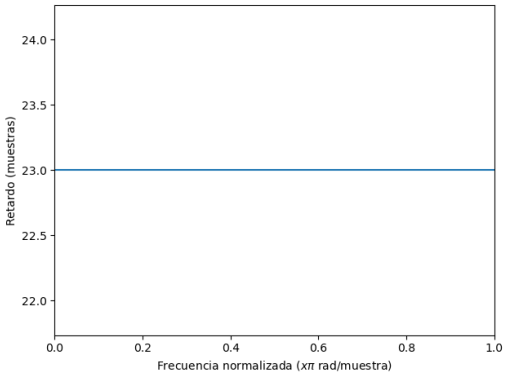


$f_c = fc + 10\,\% f_c$

Figura 5: Respuesta del sistema completo con frecuencias de corte superiores.



Respuesta del sistema completo.



Retardo

Figura 6: Uso de la función **getG722lpf** de scalib.

La **Figura 4** muestra que la respuesta en frecuencia del sistema de subbandas presenta cancelación parcial en la banda media debido a la solapación imperfecta de los filtros, con una fase de pendiente lineal (retardo constante) y una magnitud cercana a 0 dB, salvo en la

zona de transición entre subbandas, donde se observa cierta atenuación. En la **Figura 5**, al aumentar la frecuencia de corte del filtro h_1 en un 5 % y 10 %, se reduce dicha cancelación y en el caso de $f_c = fc + 10\,\%f_c$ tenemos ganancia positiva. Finalmente, la **Figura 6** muestra que el uso del filtro estándar G.722, obtenido mediante la función `getG722lpf` de `scalib`, genera una respuesta completamente plana y un retardo constante en todas las frecuencias, lo que indica una reconstrucción perfecta sin distorsión.

4. Ejercicio 4: Implementación de un codificador y descodificador según el estándar G722.

Listing 4: Codificador

```
def encoderG722(data, wlen, dataRange, bl, bh, bg):
    h1 = sc.getG722lpf() # Recupera los coeficientes almacenados del filtro paso-
        baja
    # Genera el FIR de paso alto por signo alternante, vectorizado
    signs = (-1) ** np.arange(len(h1_g722)) # Calcula el filtro paso-alta como el
         espejo del filtro paso-baja
    h2 = h1_g722 * signs
    sl = np.convolve(data, h1, mode='same') #Filtra s con el filtro h1 para
        obtener el canal de frecuencias bajas
    sh = np.convolve(data, h2, mode='same') # Filtra s con el filtro h2 para
        obtener el canal de frecuencias altas
    # Al usar mode='same', la convolucin devuelve una seal de la misma longitud
        que la seal de entrada original, manteniendo el mismo nmero de muestras.


    # Reducir la frecuencia de muestreo de las subbandas a la mitad
    sl = sl[::2] # pasos de dos (nos saltamos las impares)
    sh = sh[::2]

    # Codificar las subbandas utilizando el algoritmo APCM
    cl = encoderAPCM(sl, wlen, dataRange, bl, bg) # Codifica el canal de
        frecuencias bajas con APCM
    ch = encoderAPCM(sh, wlen, dataRange, bl, bg) # Codifica el canal de
        frecuencias altas con APCM


    # Concatenar los cdigos de las subbandas (canales)
    code = np.concatenate((cl, ch))

    return code
```

La función `encoderG722` implementa la codificación G.722 utilizando el método APCM. Primero, se obtiene el filtro pasa-baja definido en el estándar mediante la función `getG722lpf`, y a partir de él se genera el filtro paso-alta por inversión de signo alternado. Luego, se aplica la convolución del filtro paso-baja y paso-alta a la señal de entrada, dividiéndola en subbandas de bajas y altas frecuencias respectivamente. Posteriormente, se reduce la frecuencia de muestreo a la mitad en ambos canales, tomando una muestra de cada dos. Finalmente, ambas subbandas se codifican por separado usando el algoritmo APCM y sus resultados se concatenan para obtener el código final de la señal procesada.

Listing 5: Decodificador

```
def decoderG722(code, wlen, dataRange, bl, bh, bg, delay):

    # Filtros de sntesis
    h1 = sc.getG722lpf()
    k1 = 2 * h1
    n = np.arange(len(h1))
    k2 = (-1) ** (n + 1) * k1
    nLB, nHB = getBitsPerChannel(len(code), wlen, bl, bh, bg) # Num bits
        correspondiente a cada canal


    cl = code[:nLB] # Bits correspondientes al canal de frec. bajas
    ch = code[nLB:nLB + nHB] # Bits correspondientes al canal de frec. altas

    # Decodificar los bits de cada canal utilizando el algoritmo APCM
    sl = decoderAPCM(cl, wlen, dataRange, bl, bg) if bl > 0 else np.array([],
        dtype=int) # Decodifica ek canal de frecuencias bajas con APCM
    sh = decoderAPCM(ch, wlen, dataRange, bh, bg) if bh > 0 else np.array([],
        dtype=int) # Decodifica ek canal de frecuencias altas con APCM

    # np.kron hace el producto de Kronecker entre tu seal y el vector [1,0],
        insertando un 0 tras cada muestra.
    sl = sc.upsample(sl, 2) # Intercalar ceros para duplicar la frecuencia de
        muestreo del canal de frecuencis bajas
    sh = sc.upsample(sh, 2) # Intercalar ceros para duplicar la frecuencia de
        muestreo del canal de frecuencis altas


    sl_filtrada = np.convolve(sl, k1) if bl > 0 else np.array([], dtype=int) #
        filtra el canal de frec. bajas con el filtro k1
    sh_filtrada = np.convolve(sh, k2) if bh > 0 else np.array([], dtype=int) #
        filtra el canal de frec. altas con el filtro k2

    # Garantizamos que, justo antes de sumar ambas ramas, las dos seales tengan
        exactamente la misma longitud.
    #( Si no, da error, en algn momento se habrn descompensado las muestras a
        pesar de hacee la convolucin con el same)
    if bl != 0 and bh != 0:
        min_len = min(len(sl_filtrada), len(sh_filtrada))
        sl_filtrada = sl_filtrada[:min_len]
        sh_filtrada = sh_filtrada[:min_len]
        data = sl_filtrada + sh_filtrada # Combinar los dos canales para obtener
            la seal completa
    elif bl != 0:
    # Solo canal bajo existe
        data = sl_filtrada

    elif bh != 0:
    # Solo canal alto existe
        data = sh_filtrada


    return data
```

La función `decoderG722` realiza la reconstrucción de la señal original a partir del código

codificado por subbandas. En primer lugar, se calculan los filtros de síntesis: un filtro paso-baja a partir del definido por el estándar G.722, y un filtro paso-alta generado como su versión invertida con signo alternado. Luego, el código se separa en los bits correspondientes a las frecuencias bajas y altas, y se decodifican ambos canales utilizando el algoritmo APCM. Para recuperar la frecuencia de muestreo original, se intercalan ceros entre las muestras mediante el producto de Kronecker. A continuación, cada subbanda se filtra con su respectivo filtro (bajo o alto), y se ajusta su longitud para garantizar que ambas señales puedan sumarse correctamente. Finalmente, se suman las dos componentes filtradas para obtener la señal de salida reconstruida.

5. Ejercicio 5: Comparación de G722 con PCM adaptable.

Ahora, se compara un codificador por subbandas utilizando una ventana de 120 muestras y una cuantificación de 16 bits para las ganancias. Los 6 bits asignados a los datos fueron distribuidos entre los dos canales de manera variable. Para cada configuración se codificó y descodificó la señal de audio `marlene.wav`, calculando la relación señal-ruido resultante.

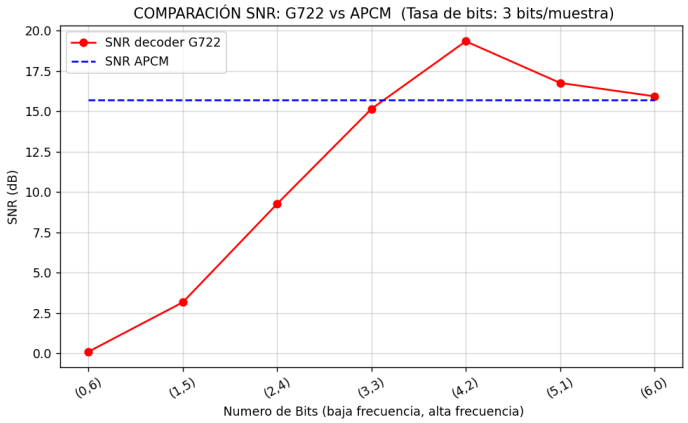


Figura 7: Comparación del algoritmo G722 con PCM adaptable.

Los resultados indican que una distribución de bits que de más importancia a las bajas frecuencias,proporciona una SNR significativamente superior a la del codificador APCM, con un valor máximo alcanzado en la configuración 4-2. Por otro lado, cabría esperar que para la configuración de (3,3) la SNR fuese igual a la de APCM, sin embargo debemos tener en cuenta que en G722 estamos cuantificando dos veces y luego sumamos, frente a la única cuantificación de APCM, esto intruduce un error ligeramente mayor y por tanto la SNR de G722 es ligeramente menor que la de APCM.

6. Conclusiones

Esta práctica ha demostrado la eficacia de las técnicas de codificación adaptativa y por subbandas en la mejora de la eficiencia en la representación digital de señales. El uso de codificadores APCM permitió observar una mejora significativa en la relación señal-ruido (SNR) respecto al PCM tradicional, gracias a la adaptación dinámica de la cuantificación.

Asimismo, el análisis de los filtros QMF mostró la importancia de un diseño preciso para evitar distorsiones en la reconstrucción. La implementación del estándar G.722, basado en estas técnicas, ofreció mejores resultados en SNR al distribuir inteligentemente los bits entre subbandas, confirmando que una asignación correcta de bits maximiza la calidad de la señal reconstruida.