Cuantificación vectorial (continuación)

Procesado de sonido

Universidad de Vigo

Diseño de un VQ : binary splitting

1. Estima un VQ con un único centroide, el correspondiente a todo el conjunto de entrenamiento.
2. Duplica el tamaño del VQ, creando a partir de cada centroide actual, $\mathbf{y}$, los vectores:

$$\begin{aligned} \boldsymbol{y}_+ &= \boldsymbol{y} \cdot (1+\varepsilon) \\ \boldsymbol{y}_- &= \boldsymbol{y} \cdot (1-\varepsilon) \end{aligned}$$

$$0 \leq \varepsilon \leq 0{,}05$$

3. Utiliza el algoritmo K-means para estimar los centroides a partir de los centroides VQini generados en el paso anterior.
4. Repite los pasos 2 y 3 hasta obtener el tamaño de código deseado.

Tarea 1: diseño de un VQ mediante binary splitting

Comprender: **function [VQ vDist] = bsVQ(data, nbits, epsilon, threshold, display)**

```
% VQ: Matriz con los centroides resultantes por filas.
% vDist: vector con la distorsión (MSE) en cada iteración de la última
%        llamada a Kmeans.
%
% data : matriz con vector de datos por fila.
% nbits: numero de bits del VQ deseado.
% epsilon: parametro para dividir cada centroide en 2 (ej: 0.025).
% threshold: umbral de parada para K-means (ej:0.01)
% display: si vale 1 y la dimensión es 2 entonces representa gráficas ilustrativas.
%
% Ejemplo: load traindata; [VQ, vDist]=bsVQ(traindata,4,0.025,1e-3,1);
```

Con los datos en *traindata.mat*:

- Diseñar un VQ de 6 bits, utilizando un umbral de parada de 0,001 y $\epsilon = 0{,}025$.
- display=1 para representar los datos de partida y los centroides resultantes.
- Representa cómo varía la distorsión obtenida, *vDist*, en función de la iteración.
- Ejecuta el algoritmo bsVQ 20 veces. Guarda el tiempo de ejecución y la distorsión de la última iteración de cada ejecución. Represéntalos gráficamente. Compara los resultados con los obtenidos para Kmeans en la sesión 1 (Ayuda: funciones Matlab *clock* y *etime*)

Tarea 2: Comparación SNR algoritmo binary splitting

Vamos a comparar la SNR obtenida con cuantificación escalar y vectorial en función de R_b.

Tarea 2a.[1] Utilizando como datos la matriz *training* (obtenida en tarea 3, sesión 1, a partir de *tvg_training_20s.wav*):

- Obtén con *bsVQ* cuantificadores vectoriales de 1 a 16 bits ($thr = 0{,}01$, $\varepsilon = 0{,}025$).
- Almacena los 16 VQs en una única matriz *VQbst* de forma que las dos primeras filas se correspondan con los centroides del VQ de 1 bit, las cuatro siguientes con los centroides del VQ de 2 bits, etc.
- Registra el tiempo que tarda en estimarse cada uno de los 16 VQs y represéntalo en función del número de bits.
- Guarda la matriz *VQbst* y los tiempos registrados en el fichero *VQbst.mat*.

Tarea 2b. Considerando la señal en *tvg_training_5s.wav* (fragmento de *tvg_training_20s.wav*):

- Representa la SNR global obtenida con los VQs en función del régimen binario. ¿Cuál es ahora el número de bits por muestra?
- Compárala con las gráficas obtenidas con un cuantificador escalar uniforme (*qmidriser*, $x_{sc} = 1$) y con las aproximaciones SNR1 y SNR2.

Tarea 2c. Repite la tarea 2b considerando la señal en *tvg_test_5s.wav*. Compara los resultados obtenidos con los de la subtarea anterior y justifica las diferencias encontradas (puede ser útil idear alguna gráfica adicional).

[1] El tiempo de ejecución puede ser largo

Tarea 2: Comparación SNR algoritmo binary splitting

Hasta el momento los datos han sido vectores de dimensión 2, pero los algoritmos diseñados son válidos para cualquier dimensión, dimensión 1 incluida.

Tarea 2d. Cuantificación escalar

- Diseñaremos un cuantificador escalar de 5 bits con la función bsVQ, utilizando como datos de entrenamiento **tvg_training_20s.wav** ($thr = 0{,}01, \varepsilon = 0{,}025$).
- De forma análoga a como se hizo para los cuantificadores uniformes, representa la característica de cuantificación del cuantificador estimado en el rango de valores de entrada de -1 a 1.
- Para la señal en *tvg_training_5s.wav* estima la SNR obtenida con el cuantificador diseñado. Sobre la gráfica obtenida en la subtarea anterior (con el mismo fichero) indica la situación del valor ahora obtenido.

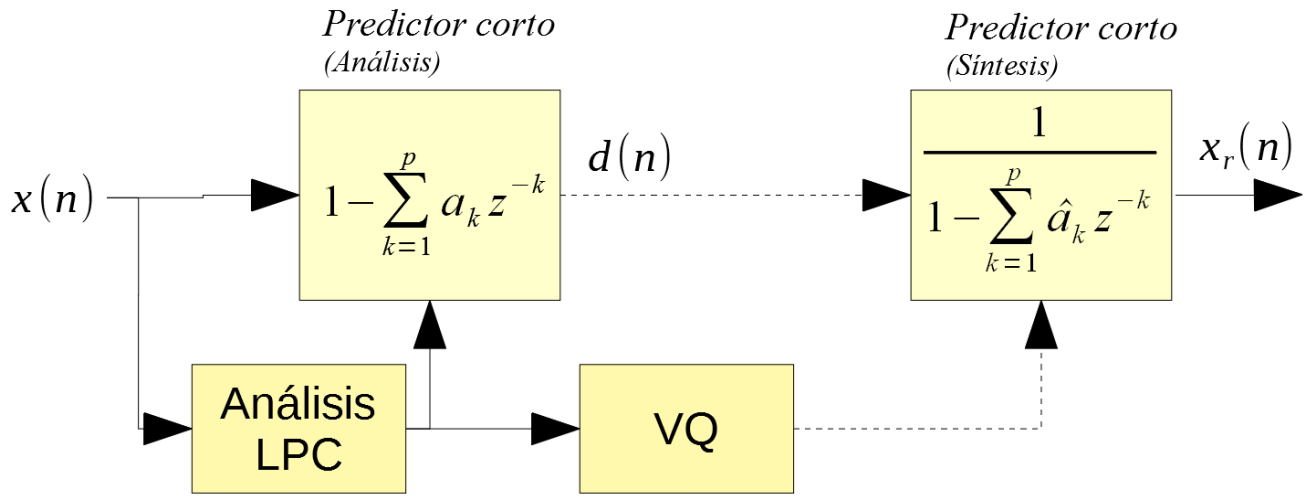
Tarea 3: Diseño de VQs para la envolvente espectral

function [LPC, Ep, RC, LSF, LAR]=speech2lpc(s, p, window, wshift)

```
% Extracción de parámmetros LPC y equivalentes de una señal.
%
% s: señal de voz o sonora.
% p: orden de predicción
% window, wshift: ventana a utilizar y su desplazamiento en muestras.
%
% Salidas: parámetros correspondientes a cada trama por filas.
% LPC, Ep: LPCs (matriz) y energía del error de predicción (vector).
% RC: coeficientes de reflexión (matriz)
% LSF: Line Spectral Frequencies (matriz)
% LAR: Log Area Ratios (matriz)
```

- Obtención de LPCs, RCs y Ep de cada trama:
  Rcorr=xcorr(frame,frame); Rcorr=Rcorr(length(frame):length(frame)+p);
  [ak,ep,rc] =levinson(Rcorr,p);
- LARs y LSFs utilizando funciones *rc2lar* y *poly2lsf*.
- Extraed los parámetros de todas las tramas del archivo *tvg_10minutos_8khz.wav* (ventana rectangular de 160 muestras, desplazamiento 160, $p = 10$). Guardadlos en el fichero *LPCpar.mat*. Este será nuestro material de entrenamiento.
- Con la función *bsVQ* ($\varepsilon = 0{,}025,\ th = 0{,}001$) entrenad VQs de 10 bits para los LPC, RC, LSF y LAR (*VQLPC, VQRC, VQLSF, VQLAR*)
- En el diseño de VQLPC no consideréis el primer coeficiente (siempre uno). Cuidado con vectores fila y columna en todos los pasos.

Tarea 4a: Análisis y síntesis LPC

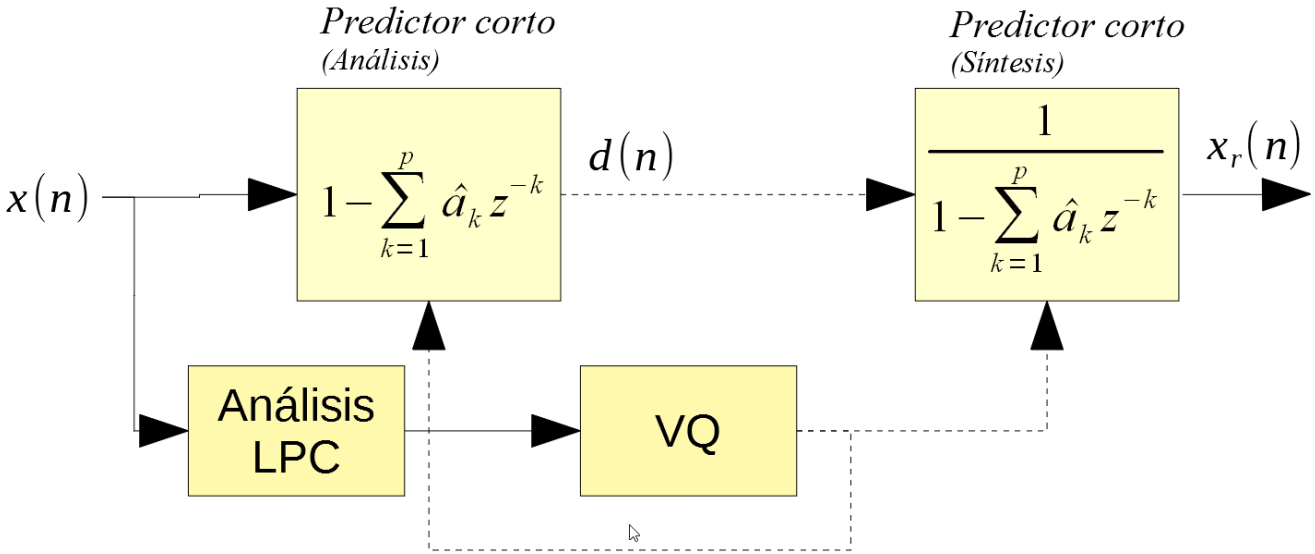


function [syn d]=syslpc_vq(s, Lframe, p, VQ, par_string)

```
% Parámetros adicionales:
% Lframe: longitud de cada trama (segmento de voz)
% VQ: matriz con los centroides por filas del VQ a aplicar.
% par_string: indica los coeficientes a cuantificar ('LPC','RC','LSF' o 'LAR').
```

- Para obtener los coeficientes $\hat{a}_k$ a partir de los parámetros cuantificados utilizad las funciones: rc2poly, lsf2poly, lar2rc.
- Para cada VQ diseñado anteriormente, y con los ficheros de *tvg_home_8khz* y *tvg_muller_8khz*, escuchad la señal reconstruída y calculad la SNR global. Utilizad Lframe=160. ¿Por cuál o cuáles de los VQs os decantaríais?.

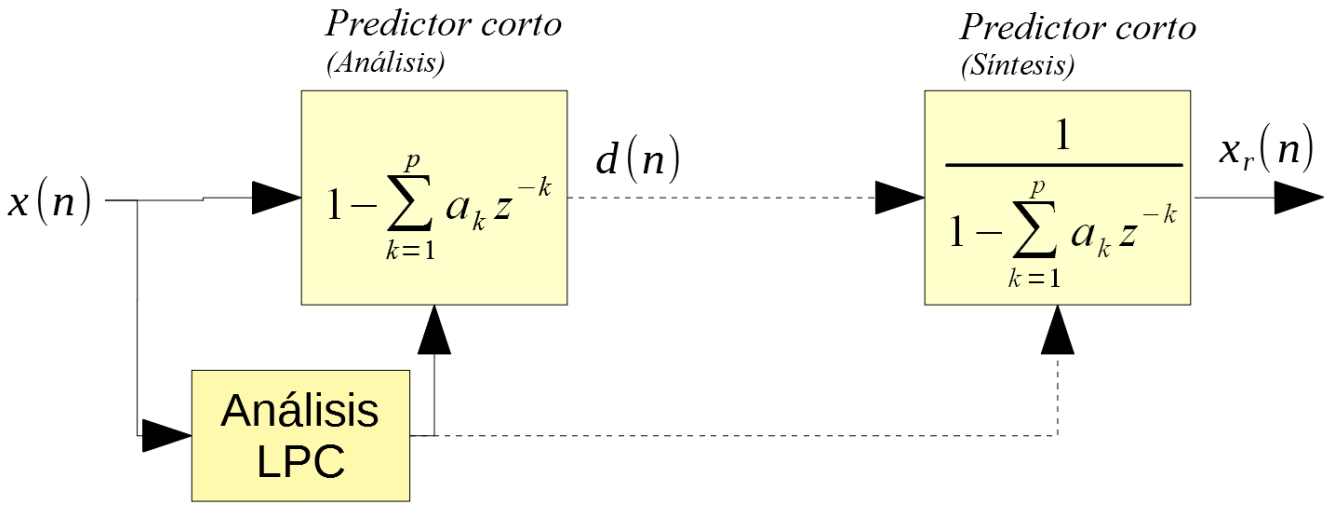
Tarea 4b: Análisis y síntesis LPC



function [syn d]=syslpc_vqq(s, Lframe, p, VQ, par_string)

- Para cada VQ diseñado anteriormente, y con los ficheros de *tvg_home_8khz* y *tvg_muller_8khz*, escuchad la señal reconstruída y calculad la SNR global. ¿Os parecen lógicos los resultados obtenidos?.

Tarea 4c: Análisis y síntesis LPC



function [syn d]=syslpc(s, Lframe, p)

```
% Análisis y reconstrucción de una señal mediante predicción lineal
% s: señal de voz.
% Lframe: longitud de la ventana rectangular empleada y del desplazamiento (no hay so
% p: orden de predicción
%
% syn: señal reconstruída
% d: secuencia d[n] (error de predicción)
```

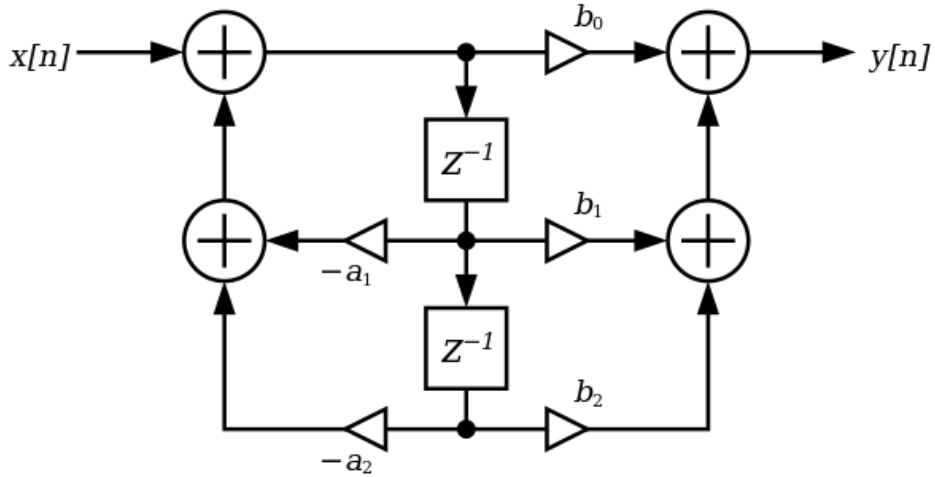
- Cuidado con las condiciones iniciales y finales de los filtros.

Tarea 4c: Análisis y síntesis LPC

- Consideraremos los ficheros *test_period_70*, *tvg_home_8khz* y *tvg_muller_8khz*. El primero es un fichero de test obtenido como respuesta de un filtro IIR a una entrada consistente en un tren de deltas con periodo 70 muestras.
- Procesaremos cada fichero completo con las funciones *syslpc* y *syslpc_vqq* (con VQLSF), y guardaremos en variables separadas las señales originales y sus respectivos errores de predicción.
- Para cada fichero representad las siguientes figuras que contendrán tres gráficas (subplot):
  - Señal *test_period_70* , error de predicción obtenido con *syslpc_vqq* y error de predicción obtenido con *syslpc* en el tramo de la muestra 1000 a la 2000.
  - Lo mismo para *tvg_home_8khz*, representando el tramo 13500:14500.
  - Repetid para *tvg_muller_8khz* en el tramo 8500:9500.
- Comentad razonadamente los resultados obtenidos en cada una de las gráficas. ¿Cómo afecta la cuantificación de los coeficientes de los filtros?

Condiciones iniciales y finales de los filtros

- Los filtros digitales contienen elementos de retardo donde se almacenan valores anteriormente calculados.
- En nuestro caso los dos filtros son de orden p y tendrán entonces p retardos.
- Las condiciones iniciales son los valores inicialmente almacenados en los retardos.
- Las condiciones finales son los valores que quedan almacenados al terminar de procesar un segmento.
- Al procesar sucesivos segmentos las condiciones iniciales del filtro deben ser las finales del segmento anterior (nulas si se trata del primer segmento).
- Cada filtro tiene sus propias condiciones iniciales y finales.



Ejemplo: Si b es un vector con los coeficientes de un filtro FIR y x es la entrada

$$[y, zi] = filter(b, 1, x, zi);$$

filtra la señal considerando el vector zi como condiciones iniciales y devolviendo las condiciones finales en el mismo vector.