Práctica 1: Codificación de voz

Introducción y objetivos

Esta práctica tiene como objetivo general la familiarización del alumno con las características específicas de la señal de voz que se explotan para su codificación eficiente.

Los objetivos de la práctica son:

- Adquirir conocimientos de los parámetros fundamentales de la señal de voz y la capacidad para extraerlos.
- Experimentar su uso para codificación de voz.

Guarde y documente todo el código que genere pues le será de utilidad para responder a las cuestiones planteadas al final de la práctica.

Al comienzo de la práctica se proporcionará código de ejemplo que puede optar por utilizar o no. Conforme vaya progresando irá teniendo más libertad para programar su propio código.

1 Efectos de la ventana sobre la señal de voz

En primer lugar estudiaremos cómo afecta el enventanado a la señal de voz. Cargue el fichero `act_1A.wav` en Matlab usando la función `audioread`. La voz está grabada utilizando una frecuencia de muestreo de 8 KHz. A continuación utilice la función `stft.m` proporcionada con el material de esta práctica para realizar las siguientes tareas:

- Utilice la función `hamming` de Matlab para obtener una ventana de 256 muestras. Represente la ventana en una gráfica y observe su forma.
- Obtenga una ventana Hamming de 25 ms de duración. Llame a la función `stft` para analizar las tramas de la señal de voz usando la ventana que ha obtenido, FFT de 1024 muestras y solapamiento de media ventana. Recuerde que estos dos últimos valores se deben entregar a la función pasados a número de muestras.
- Como puede comprobar, la función permite analizar lo que le ocurre a las tramas antes y después de multiplicarse por la ventana, en tiempo y en frecuencia. Amplíe la figura para

verlo mejor. ¿Qué efecto tiene la aplicación de la ventana sobre la trama en el dominio temporal? Intente encontrar alguna ventaja.

- Comente el efecto en frecuencia que tiene la multiplicación por la ventana en tiempo. ¿Puede relacionarlo con alguna propiedad de la DFT? Pruebe con más tamaños de ventana para tener más información.

2 Tipos de fuente de producción de voz

A continuación analizaremos una señal de voz más corta con calidad telefónica (muestreada a 8000 Hz), `prueba8.wav`.

Localizaremos en ella tramos de voz sonoros y sordos.

- Dibuje la señal de voz en el dominio temporal y localice un tramo sonoro.
- Represente ahora su espectro utilizando la DFT. Utilice el código que se detalla a continuación (asegúrese de comprende totalmente cada línea de código):

```
% x es una señal de voz, N es la longitud de la trama (debe ser par),
% S es la primera muestra de la señal que se evalúa
xf = x(S:S+N-1).*hanning(N);
X = fft(xf);
figure(1); clf;
plot(10*log10(abs(X(1:N/2+1)).^2));
```

- ¿A qué frecuencia (en Hz) corresponde el valor de X(5)?
- ¿Cuál es la frecuencia fundamental (en Hz) de la señal que está analizando? Calcúlelo tanto en el dominio temporal como en el frecuencial y asegúrese de que coincide razonablemente.
- ¿Qué longitudes de trama N son más adecuados para este cálculo?
- Ahora que ha reflexionado sobre qué frecuencias corresponden a cada muestra de la FFT, puede representar los ejes de las gráficas con sus verdaderos valores. A partir de ahora, pase a segundos todos los ejes temporales y a hercios todos los ejes frecuenciales de sus graficas.
- Extraiga la envolvente LP añadiendo al anterior código lo siguiente (asegúrese de comprender cada línea de código identificando las sentencias con las expresiones matemáticas de la teoría).

```
c = xcorr(xf, xf, M); % M es el orden de predicción.
[a, e]= levinson(c(M+1:2*M+1));
% a es un vector cuyo primer elemento es siempre 1.
a = a(:); % Necesitamos que sea un vector columna
% El código comentado a continuación de esencialmente
el mismo resultado que las dos líneas anteriores
% r = c(M+1:2*M+1);
% R = toeplitz(r(1:M),r(1:M)); % Una matriz Toeplitz con filas r(1:M)
% a = - R / r(2:M+1);
% a = [1; a]; % Añadir el 1 al comienzo del vector
% e = sum(a*r);
h = zeros(N,1);
for k=0:N-1
h(k+1) = 1 / (a'*exp(-i*2*pi*k/N*(0:M)') );
end
% Un equivalente más rápido al bucle anterior:
% h = freqz(1, a, N, 'whole');
hold on
plot(10*log10(e*abs(h(1:N/2+1)).^2), 'r');
```

- ¿Qué orden de predicción es más adecuado para representar la envolvente espectral?
- Repita lo anterior para una trama sorda.

3 El espectrograma

A continuación programaremos un espectrograma en Matlab (cuando acabe puede comparar el suyo con el existente en Matlab (`spectrogram`)

Un espectrograma muestra la energía de la señal de voz como una función del tiempo y la frecuencia. El resultado se representa normalmente con el tiempo en el eje horizontal, la frecuencia en el vertical y clásicamente se representa la energía (en dB) en una escala de grises donde el negro corresponde con energías altas y el blanco con las bajas.

Por lo tanto necesitamos recorrer la señal de voz haciendo análisis de Fourier en ventanas sucesivas. Habitualmente dichas ventanas se solapan para evitar cambios muy bruscos entre tramas consecutivas. En la Figura 3 se representan los dos elementos que dirigen el flujo del análisis.

En esta sección, utilice la señal corta de la sección anterior para sus pruebas.

- En primer lugar programe una función que extraiga las tramas solapadas y las represente en el eje temporal. Puede añadir, si quiere, el comando sound en cada una de ellas para escuchar la evolución del análisis. Utilice el siguiente código si lo considera oportuno. Invoque `spectrogram(voz, 256, 32)` y observe la animación.

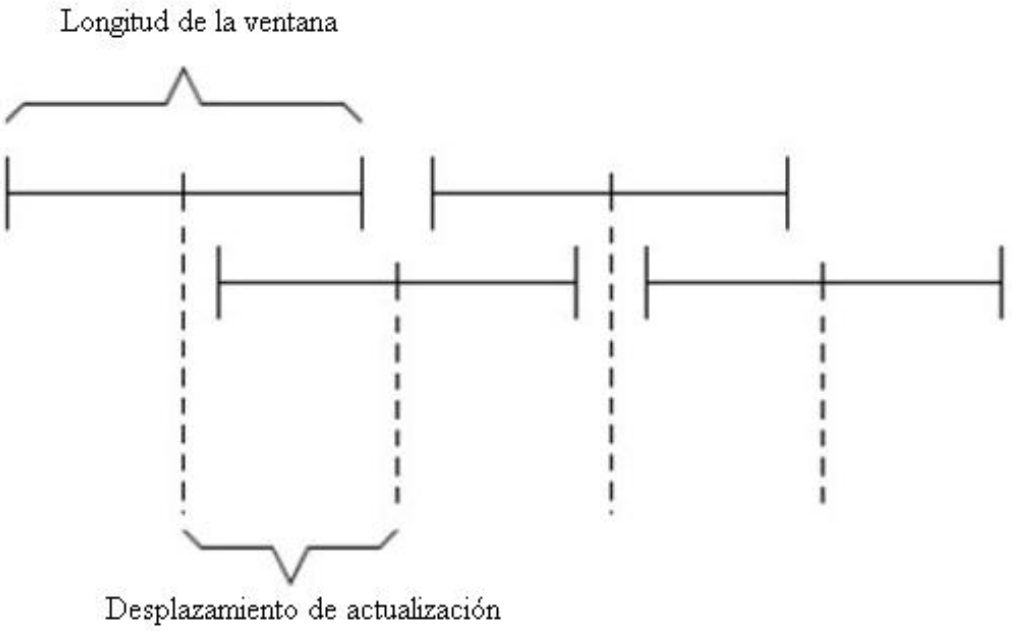

Figure 1: Longitud y desplazamiento de la ventana de análisis

```
function CIMC_spectrogram(x, alen, ulen
% x es la señal de voz
% alen es la longitud de la ventana de análisis
% ulen es el desplazamiento de actualización
N = length(x);
naf = floor((N-alen+ulen)/ulen); % Número de ventanas
n1 = 1;
n2 = alen;
for n=1:naf % Contador sobre el número de ventana
xf = x(n1:n2);
figure(1); clf; plot(xf); axis([1 alen min(x) max(x)]); pause(0.05)
n1 = n1 + ulen;
n2 = n2 + ulen;
end
```

- Ahora añada el análisis frecuencial usando el código que generó en la sección anterior. No se olvide de utilizar una ventana de hanning antes de hacerlo. Para mostrar simultáneamente el análisis temporal y frecuencial utilice subplot.

- Con estos dos elementos ya podemos implementar el espectrograma clásico generando una matriz cuyas columnas sean los valores en el dominio transformado. No olvide reservar espacio para dicha matriz haciendo `S = zeros(alen/2+1, naf)` e introduzca en el bucle de análisis de su función la asignación columna a columna con `S(:,n) = ...`. Por último,

muestre la matriz S con el siguiente código:

```
colormap(gray)
% El espectrograma clásico es gris
% Puede consultar help gray para ver otros mapas de color
% Flipud hace que se invierta los niveles de gris
% de manera que el negro corresponda con altas energías
% y el blanco con bajas
imagesc(flipud(-S));
```

- En este momento ya puede comentar el código que mostraba la animación inicial. Pruebe con distintas longitudes de la ventana de análisis y observe las diferencias, ¿puede observar las trayectorias de los formantes? ¿Puede identificar tramos de voz sonoros y sordos?

4 Estimación de parámetros

En esta sección analizaremos (estimaremos) los parámetros de la señal de voz que utiliza un vocoder para pasar en la próxima sección a sintetizar la señal de voz correspondiente.

La estimación se debe hacer trama a trama así que utilizaremos el mismo esquema que en la sección anterior.

- Comencemos con la estimación de la energía de cada trama (normalizada por muestra):
  $E_X = \frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2$
  Donde `N` es la longitud de la ventana. Escriba una función que devuelva las energías en un vector. Comience con el esqueleto de la función del punto 1 de la sección anterior. No olvide crear una variable para el vector completo.

```
function E = analysis(x, alen, ulen)
% Inicialización
E = zeros(naf, 1);
% Dentro del bucle calculamos la energía para cada n
E(n) = ...
```

- Ahora debemos detectar si una trama es sonora o sorda. Esto es un problema difícil, aunque aquí lo resolveremos de forma muy sencilla contabilizando los cruces por cero de la señal de voz ya que, cuando la señal es sonora el número de cruces es menor que en el caso sordo. Extienda la función que diseñó en el punto anterior así:

```
function [E, ZC, V] = analysis(x, alen, ulen)
% Inicialización
E = zeros(naf, 1);
ZC = zeros(naf, 1);
V = zeros(naf, 1);
% Dentro del bucle
E(n) = ...
ZC(n) = ... % Número de cruces por cero normalizado por N
V(n) = ... % Decisor: si es 1 la trama es Sonora, si es 0 es sorda
```

(Recomendamos un umbral del decisor sonoro/sordo de 0.375)

- A continuación necesitamos los coeficientes de predicción lineal que calculamos en la Actividad 2. Incorpórelos a la función y guárdelos en las filas de una matriz A como sigue:

```
function [E, ZC, V, A] = analysis(x, alen, ulen, M)
% Initialization
E = zeros(naf, 1);
ZC = zeros(naf, 1);
V = zeros(naf, 1);
A = zeros(naf, M+1); % M es el orden de predicción.
% M+1 permite que guardemos el 1 inicial como en la actividad 2.
% Dentro del bucle
E(n) = ...
ZC(n) = ...
V(n) = ...
A(n,:) = ... %¡¡ Asegúrate de almacenarlo en filas!!
```

- Finalmente debemos extender el código con la estimación de la frecuencia fundamental (pitch). Este parámetro es muy crítico haciendo que mejore de forma muy importante la calidad de un codificador si se realiza una buena estimación. Por ello y a pesar de que su estimación se puede incluir en la función anterior, vamos a realizarlo con la función `pitch.m` proporcionada.

- Dibuje las salidas de su función de análisis junto a la de pitch probándolo sobre la señal `prueba8.wav` usando el siguiente código:

```
figure(1);
clf;
subplot(3,2,1)
plot(x) % Señal de entrada
axis([1 length(x) min(x) max(x)]);
subplot(3,2,2)
plot(sqrt(E)) % Desviación estándar de la energía
axis([1 length(E) min(sqrt(E)) max(sqrt(E))]);
subplot(3,2,3)
plot(V, .) % Decisión sonoro/sordo
axis([1 length(V) 0 1]);
subplot(3,2,4)
plot(ZC) % Número de cruces por cero
axis([1 length(ZC) min(ZC) max(ZC)]);
subplot(3,2,5)
F = 8000./P;
plot(F) % Frecuencia fundamental en Hz
axis([1 length(F) 0 600]);
subplot(3,2,6)
S = zeros(512, naf);
for n=1:naf
S(:,n) = 20*log10(abs(freqz(1,A(n,:),512)));
end
S = flipud(S);
colormap(gray);
imagesc(S); % Envolvente spectral en estilo espectrograma
```

5 El vocoder

En esta sección sintetizaremos la señal de voz a partir de los parámetros extraídos en la sección anterior. Esta es una labor compleja en la que hay que tener muy en cuenta los efectos de borde en cada una de las tramas. Por ese motivo utilizaremos aquí una función ya preparada `synthesis.m`. Además, para completar verdaderamente un vocoder deberíamos añadir aquí un cuantificador de los parámetros pero no entraremos en ello en esta práctica.

- Sintetice la señal de voz utilizando:

```
function [x] = synthesis(E, V, A, P, U)
% [x] = synthesis(E, V, A, P, U) sintetiza voz basándose en los parámetros
% E, P, V y A, y devuelve las muestras sintéticas en la variable x.
% U es la longitud de actualización.
```

- Asegúrese de que la señal sintetizada es correcta. Si recupera un mensaje inteligible, verificará el buen funcionamiento de su codificador.
- ¿Cómo calificaría a la señal sintética en una escala DMOS?