Práctica 4
LPC

Anastópulos Matías - 95120

5 de abril de 2018

Índice

Resumen

El objetivo del presente trabajo es implementar el calculo de los coeficientes LPC con el fin de realizar un análisis sobre una pista de audio. Mediante dicho análisis se podrá obtener los formantes que permitirán identificar los distintos fonemas.

1. LPC Y Envolvente Del Espectro

1.1. LPC En Ventana Única

El objetivo en esta parte del trabajo es poder realizar el calculo de los coeficientes LPC para una ventana fija que contenga un único fonema. Se utilizaran los coeficientes para poder predecir la señal de audio, y se realizará una comparación entre ambas. Se analizará la magnitud del error cometido y su naturaleza. Finalmente se utilizarán estos coeficientes para poder estimar la envolvente del espectro, que contiene información sobre los formantes y por lo tanto de la posición de las distintas partes del tracto vocal.

1.1.1. Código

El siguiente código es el utilizado para obtener lo anteriormente dicho.

```
close all;
clear;

tiempo_ventana = 0.025;
archivo = 'fantasia.wav';
M = 20;

%Cargo el archivo.
[audio, fs] = audioread(archivo);

tamanio_ventana = tiempo_ventana * fs;

%Obtengo la porcion donde esta la 'a'.
audio_ventaneado = audio(14000 : (14000 + tamanio_ventana - 1));

%Calculo de coeficientes LPC.
r = xcorr(audio_ventaneado);
r = r(tamanio_ventana : (tamanio_ventana + M - 1));
R = toeplitz(r(1: M - 1));

b = R\r(2 : M);
G = sqrt(r(1) - b' * r(2 : M));

%Estimacion.
audio_estimado = filter([0; b], 1 , audio_ventaneado);

%Grafico señal estimada vs señal original.
figure(1);
hold on;
tiempos = linspace(0, tamanio_ventana / fs, tamanio_ventana);
plot(tiempos, audio_ventaneado);
plot(tiempos, audio_estimado);
title('Señal Estimada VS Original.');
legend('Original', 'Estimada');
xlabel('Tiempo (s)');
ylabel('Amplitud');

%Grafico error.
figure(2);
plot(tiempos, audio_ventaneado - audio_estimado);
title('Error De Estimación.');
legend('Error');
xlabel('Tiempo (s)');
ylabel('Amplitud');

%Grafico espectro vs envolvente.
figure(3);
hold on;
frecuencias = linspace(0, fs*((tamanio_ventana-1)/tamanio_ventana), tamanio_ventana);

plot(frecuencias, abs(fft(audio_ventaneado)));

envolvente = freqz(G, [1; -b], tamanio_ventana, 'whole');
plot(frecuencias, abs(envolvente));

axis([0, fs/2, 0, 25]);

title('Espectro VS Envolvente.');
legend('Espectro', 'Envolvente');
xlabel('Frecuencia (Hz)');
ylabel('Amplitud');
```

Codigo/ejercicio_1_unica_ventana.m

1.1.2. Señal Original VS Estimada

En la figura 1 podemos ver la comparación de la señal original y la estimada. La diferencia se podrá analizar mejor en el gráfico del error en la siguiente sección.

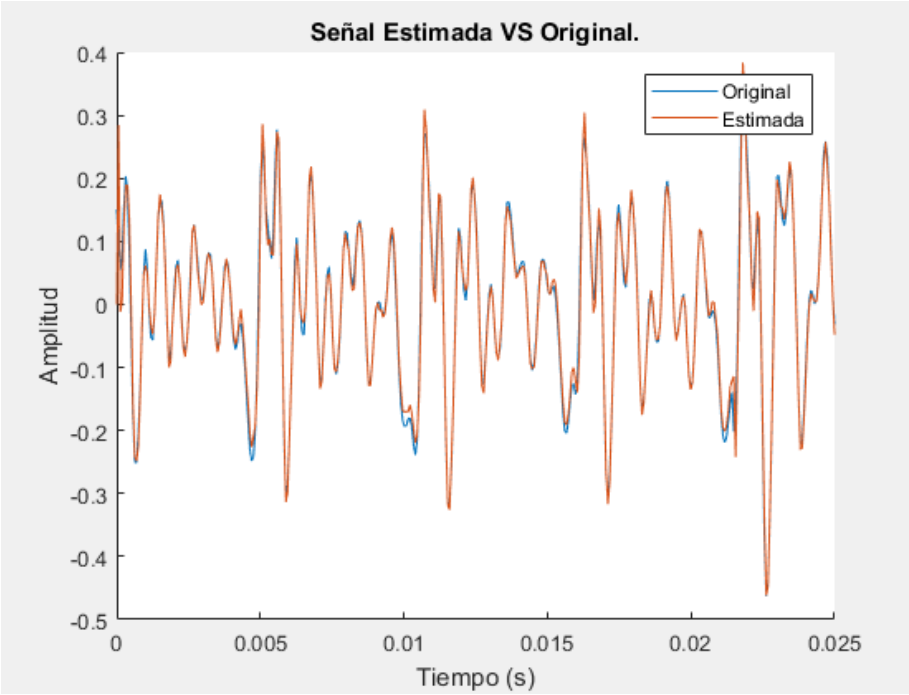

Figura 1: Señal Original VS Señal Estimada

1.1.3. Error De Estimación

En la figura 2 podemos ver el error de estimación, es decir, la parte de la señal que no pudimos predecir. Puede verse que tiene cierta periodicidad cada 5ms, esto esta relacionado con la frecuencia glótica. Dado a que en la teoría se supuso que la entrada era ruido blanco, es lógico esperar un patrón de error de la forma del pulso glótico, ya que este no lo es. Por otro lado puede verse que la amplitud del error no supera los 0,08.

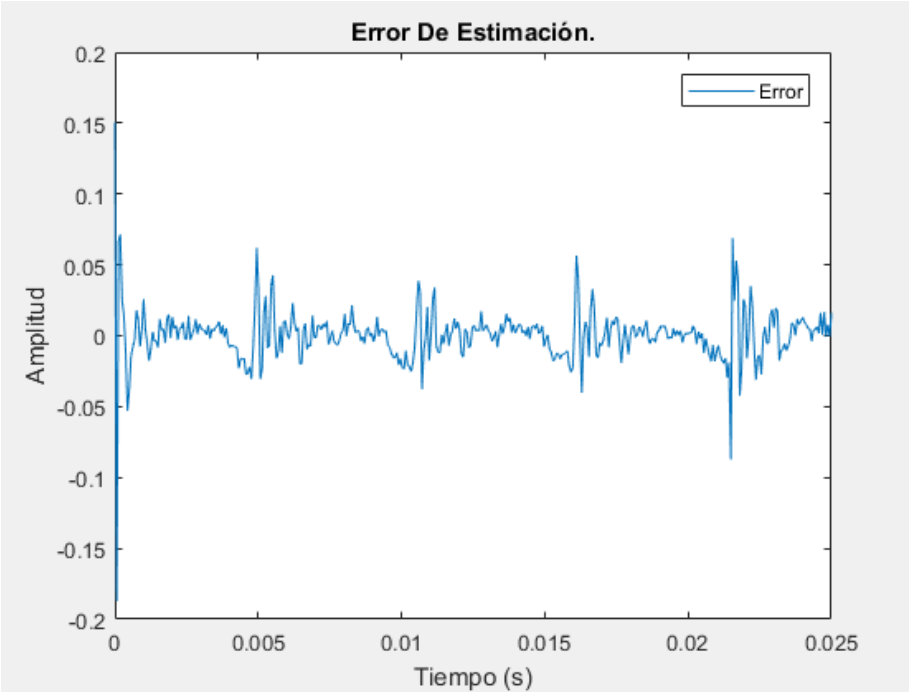

Figura 2: Error De Estimación

1.1.4. Espectro VS Envolvente

En la figura 3 podemos ver una comparación entre el espectro de la señal original y la envolvente obtenida mediante los coeficientes LPC. Puede verse el pico principal cerca de los 720Hz.

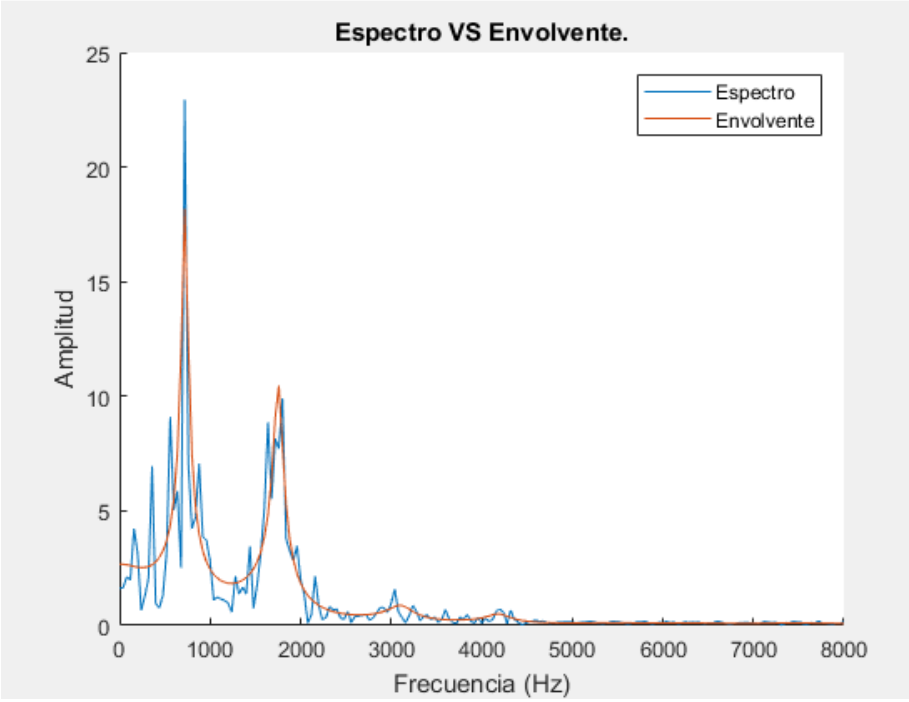

Figura 3: Espectro VS Envolvente

1.2. Envolventes De Las Vocales

Esta sección del trabajo esta destinada a poder obtener las envolventes de las distintas vocales que aparecen en el audio. Se presentarán las mismas en un solo gráfico para su comparación.

1.2.1. Código

El siguiente código es el utilizado para obtener lo anteriormente dicho.

```
close all;
clear;

tiempo_ventana = 0.025;
tiempo_avance_ventana = 0.010;
archivo = 'fantasia.wav';
M = 20;

%Cargo Audio.
[audio, fs] = audioread(archivo);

tamanio_ventana = tiempo_ventana * fs;

%Recorto las partes vacias.
inicio_de_audio = 9000;
fin_de_audio = 25000;
audio = audio(inicio_de_audio : fin_de_audio);

%Grafico envolventes para cada vocal.
posicion_vocales = [1500, 5000, 9000, 12500];

figure(1);
hold on;

frecuencias = linspace(0, fs*((tamanio_ventana-1)/tamanio_ventana), tamanio_ventana);

for posicion_vocal = posicion_vocales
```

```
    audio_ventaneado = audio(posicion_vocal : (posicion_vocal + tamanio_ventana - 1));
    r = xcorr(audio_ventaneado);
    r = r(tamanio_ventana : (tamanio_ventana + M - 1));
    R = toeplitz(r(1: M - 1));
    b = R\r(2 : M);
    G = sqrt(r(1) - b' * r(2 : M));
    envolvente = abs(freqz(G, [1; -b], tamanio_ventana, 'whole'));

    plot(frecuencias, envolvente);
end

axis([0, fs/2, 0, 20]);

title('Envolventes De Vocales.');
legend('a', 'a', 'i', 'a');
xlabel('Frecuencia (Hz)');
ylabel('Amplitud');
```

Codigo/ejercicio_1_vocales.m

1.2.2. Envolventes De Las Vocales

En la figura 4 podemos ver las envolventes para las distintas vocales. Puede verse los tres formantes para cada una de ellos. Entre las distintas [a] no hay mayor diferencia, salvo por su amplitud. Por otro lado la única que se distingue es la [i], que tiene el primer formante a una frecuencia menor que las [a], y el segundo y tercer formante a frecuencias superiores.

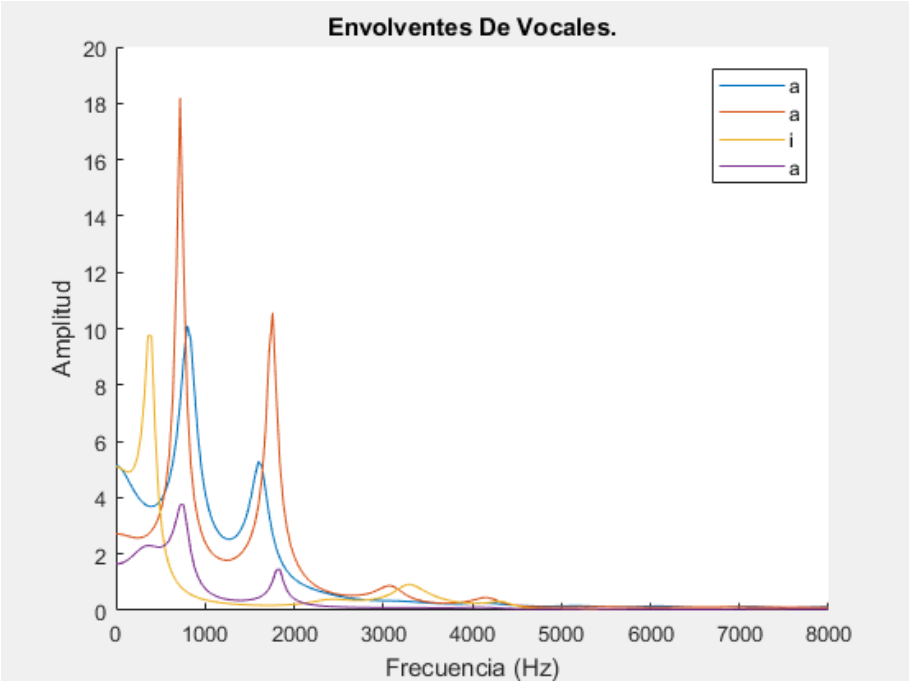

Figura 4: Envolventes De Las Vocales

1.3. Envolvente De Todo El Audio VS Espectrograma

Esta sección pretende generar el espectrograma y las envolventes de todo el audio, ventaneado cada 10ms con ventanas de un tamaño de 25ms.

1.3.1. Código

El siguiente código es el utilizado para obtener lo anteriormente dicho.

```
close all;
clear;
```

```
tiempo_ventana = 0.025;
tiempo_avance_ventana = 0.010;
archivo = 'fantasia.wav';
M = 20;

%Cargo el audio.
[audio, fs] = audioread(archivo);

tamanio_ventana = tiempo_ventana * fs;
avance_ventana = tiempo_avance_ventana * fs;

%Recorto las partes vacias.
inicio_de_audio = 9000;
fin_de_audio = 25000;
audio = audio(inicio_de_audio : fin_de_audio);

posicion_ventana = 1;
i = 1;
envolvente = zeros(ceil(length(audio) / avance_ventana), tamanio_ventana);

%Obtengo coeficientes LPC para cada ventana.
while (posicion_ventana + tamanio_ventana < length(audio))
    audio_ventaneado = audio(posicion_ventana : (posicion_ventana + tamanio_ventana - 1))
    ;
    r = xcorr(audio_ventaneado);
    r = r(tamanio_ventana : (tamanio_ventana + M - 1));
    R = toeplitz(r(1: M - 1));
    b = R\r(2 : M);
    G = sqrt(r(1) - b' * r(2 : M));

    envolvente(i,:) = abs(freqz(G, [1; -b], tamanio_ventana, 'whole'));

    i = i + 1;
    posicion_ventana = posicion_ventana + avance_ventana;
end

%Grafico de envolventes.
figure(1);
[a,b] = size(envolvente);
tiempos = linspace(0, length(audio) / fs, a);
frecuencias = linspace(0, fs*((tamanio_ventana-1)/tamanio_ventana), tamanio_ventana);
s  = surf(frecuencias, tiempos, 10*log(envolvente));
s.EdgeColor = 'none';
view(2);
axis([0, fs / 2, 0, length(audio) / fs])

title('Envolventes.');
xlabel('Frecuencia (Hz)');
ylabel('Tiempo (s)');

%Grafico De Espectrograma.
figure(2);
spectrogram(audio, hamming(tamanio_ventana),(tamanio_ventana - avance_ventana),
    tamanio_ventana, fs);
title('Espectrograma.');
```

Codigo/ejercicio_1_audio_completo.m

1.3.2. Envolventes VS Espectrograma

En las figuras 5 y 6 podemos ver las envolventes y el espectrograma del audio completo. Puede verse que ambos tienen contenido en frecuencias similares y tiempos similares, pero el espectrograma contiene mas armónicos y el gráfico de envolventes es un poco mas suave. Pueden observarse en el diagrama de envolventes las 4 vocales, que es donde mas claro se ven los formantes. Por otro lado en la transición de la [i] a la [a] del final no es abrupta sino que es continua.

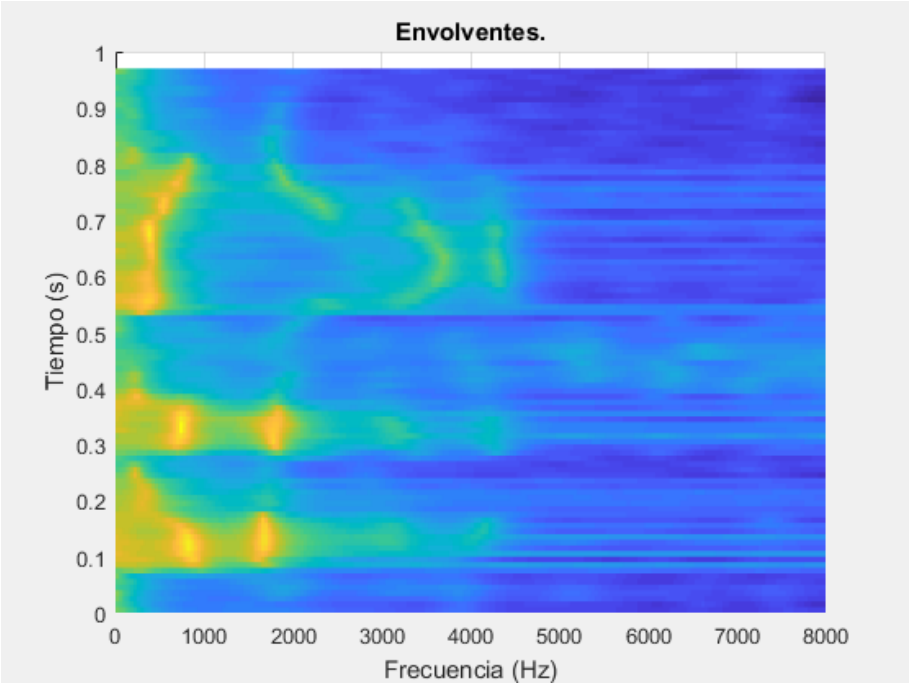

Figura 5: Envolventes Del Audio Completo

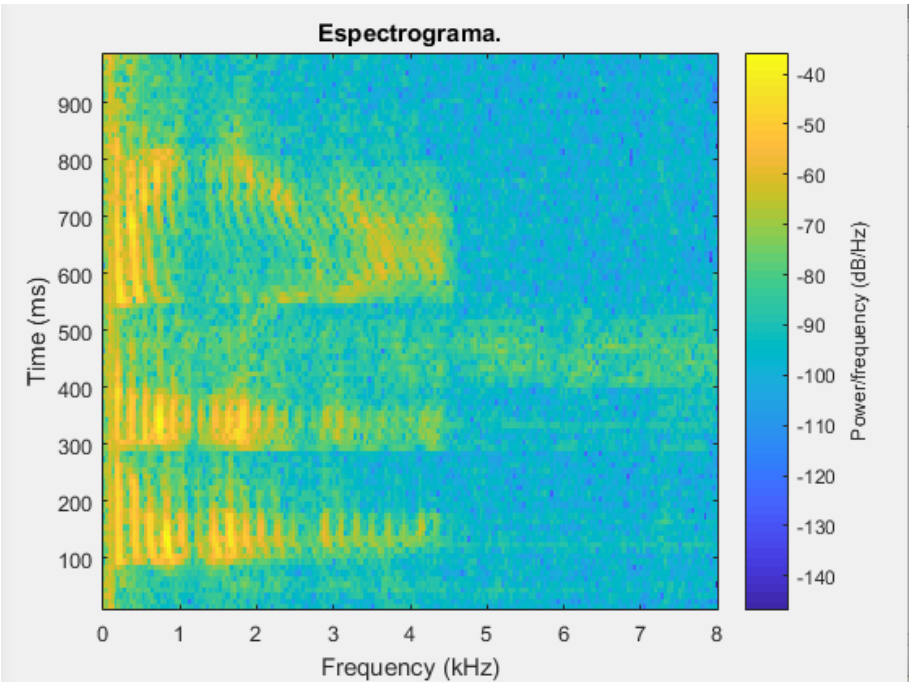

Figura 6: Espectrograma Audio Completo

2. LPC Aplicado A Codificación

2.1. LPC Y Codificación

El objetivo de esta sección es mostrar como los coeficientes LPC sirven para la codificación de una señal de voz. Se buscará, a partir de la señal de audio, generar una codificación y posteriormente una reconstrucción con diferentes grados de precisión. La codificación consistirá en representar a la señal mediante los coeficientes LPC y la señal de error de estimación. Mediante el error y los coeficientes

reconstruiremos la señal original. Primero presentaremos la codificación y reconstrucción de la señal, codificando el error en punto flotante de doble precisión.

2.1.1. Código

El siguiente código es el utilizado para obtener lo anteriormente dicho. El código además genera un archivo 'reconstruida.wav' en un formato audible con la reconstrucción de la señal.

```
close all;
clear;

tiempo_ventana = 0.025;
tiempo_avance_ventana = 0.010;
archivo = 'fantasia.wav';
M = 20;

%Cargo el audio.
[audio, fs] = audioread(archivo);

tamanio_ventana = tiempo_ventana * fs;
avance_ventana = tiempo_avance_ventana * fs;

%Recorto las partes vacias.
inicio_de_audio = 9000;
fin_de_audio = 25000;
audio = audio(inicio_de_audio : fin_de_audio);

%---- Transmisor ----

posicion_ventana = 1;
i = 1;
b_matriz = zeros(ceil(length(audio) / avance_ventana), M - 1);
error_total = [];
z = zeros(M - 1, 1);

while (posicion_ventana + tamanio_ventana < length(audio))
    %Hallo los coeficientes.
    audio_ventaneado = audio(posicion_ventana : (posicion_ventana + tamanio_ventana - 1))
    ;
    r = xcorr(audio_ventaneado);
    r = r(tamanio_ventana : (tamanio_ventana + M - 1));
    R = toeplitz(r(1: M - 1));
    b = R\r(2 : M);
    G = sqrt(r(1) - b' * r(2 : M));

    %Guardo los coeficientes.
    b_matriz(i,:) = b;

    %Guardo Error.
    [error, z] = filter([-1; b], 1, audio_ventaneado(1 : avance_ventana), z);
    error_total = [error_total; error];

    i = i + 1;
    posicion_ventana = posicion_ventana + avance_ventana;
end

%---- Receptor ----

%Debe reconstruir la señal a partir de b_matriz y error_total.

posicion_ventana = 1;
i = 1;
z = zeros(M - 1, 1);
audio_reconstruido_total = [];

while (posicion_ventana + tamanio_ventana < length(error_total))
    %Tomo una ventana del error.
    error_ventaneado = error_total(posicion_ventana : (posicion_ventana + tamanio_ventana
    - 1));

    %Filtro Inverso
    [audio_reconstruido, z] = filter(1, [-1; b_matriz(i,:)'], error_ventaneado(1:
    avance_ventana), z);
```

```
64      %Agrego al total.
        audio_reconstruido_total = [audio_reconstruido_total; audio_reconstruido];
66
        i = i + 1;
68      posicion_ventana = posicion_ventana + avance_ventana;
    end
70
    %Genero Audio Reconstruido.
72  audiowrite('reconstruida.wav', audio_reconstruido_total, fs);

74  %Grafico Señal Original VS Reconstruida Para Una Parte.
    figure(1);
76  hold on;

78  tiempos = linspace(0, length(audio) / fs, length(audio));
    plot(tiempos, audio);
80  tiempos = linspace(0, length(audio_reconstruido_total) / fs, length(
        audio_reconstruido_total));
    plot(tiempos, audio_reconstruido_total, '--');
82
    title('Señal Original VS Reconstruida.');
84  legend('Original', 'Reconstruida');
    xlabel('Tiempo (s)');
86  ylabel('Amplitud');
    axis([0.156, 0.174, -0.4, 0.4]);
```

Codigo/ejercicio_2_sin_redondeo.m

2.1.2. Señal Original VS Reconstruida

En la figura 7 podemos ver la comparación de la señal original y la reconstruida, para una pequeña porción de la señal.

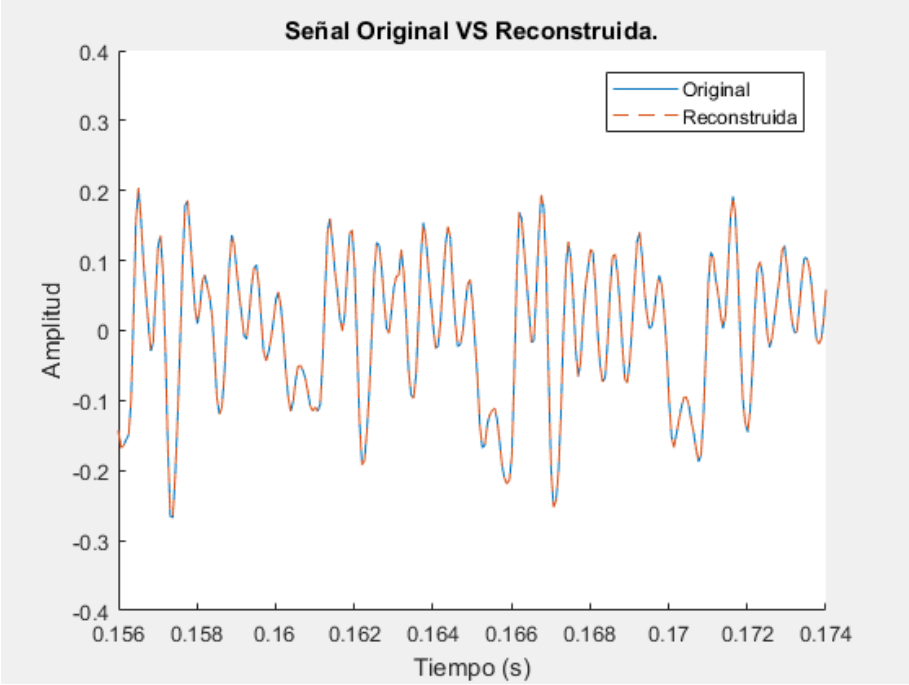

Figura 7: Señal Original VS Reconstruida

2.2. LPC Y Codificación Con Truncamiento

En esta sección pretendemos realizar lo mismo que en el caso anterior pero truncar la señal de error a una cantidad de bits determinada de precisión. Para eso se realizó una función que lo que realiza es simular el truncamiento de la señal a una cierta cantidad de bits. Dicha función es la siguiente:

```
function senial_truncada = truncar(senial, bits)

    minimo = min(senial);
    maximo = max(senial);

    %La hago toda positiva.
    senial = senial + abs(minimo);

    %La divido en 2^bits niveles.
    senial = round(senial * (2^bits - 1) / (maximo + abs(minimo)));

    %La vuelvo a pasar a punto flotante.
    senial = senial * (maximo + abs(minimo)) / (2^bits - 1);

    %Le saco el nivel de continua.
    senial_truncada = senial - abs(minimo);
end
```

Codigo/truncar.m

En la figura 8 podemos ver el uso de esta función para redondear una señal senoidal a 3 bits.

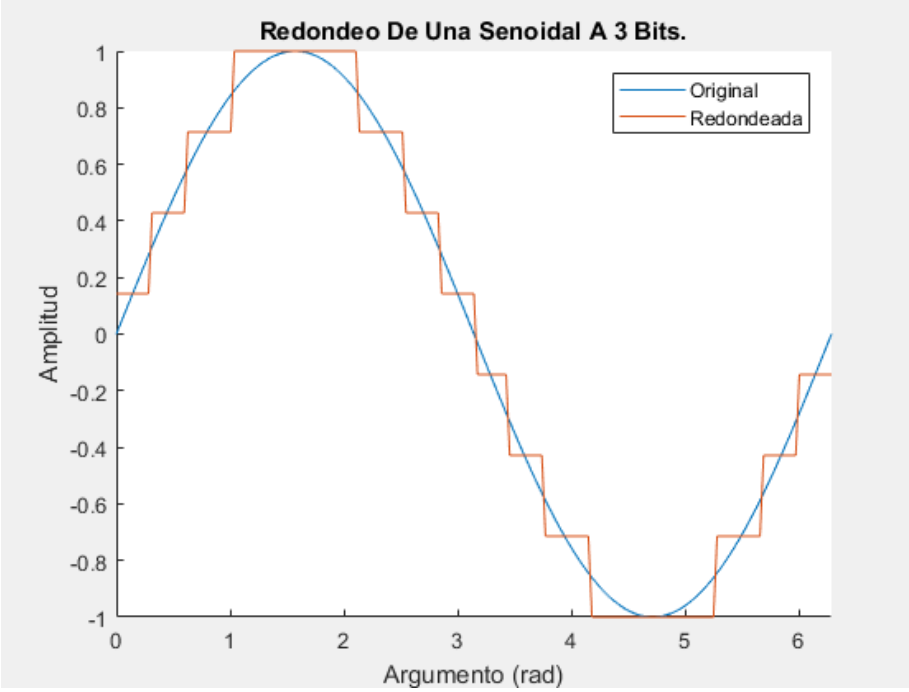

Figura 8: Redondeo De Una Senoidal A 3 Bits

2.2.1. Código

El siguiente código realiza lo mismo que el caso anterior, solo que redondeando la señal de error a 4, 6, 8 y 16 bits, simulando el error de cuantización.

```
close all;
clear;

tiempo_ventana = 0.025;
tiempo_avance_ventana = 0.010;
archivo = 'fantasia.wav';
M = 20;

%Cargo el audio.
[audio, fs] = audioread(archivo);

tamanio_ventana = tiempo_ventana * fs;
avance_ventana = tiempo_avance_ventana * fs;
```

```
%Recorto las partes vacias.
inicio_de_audio = 9000;
fin_de_audio = 25000;
audio = audio(inicio_de_audio : fin_de_audio);

%--- Transmisor ----

posicion_ventana = 1;
i = 1;
b_matriz = zeros(ceil(length(audio) / avance_ventana), M - 1);
error_total = [];
z = zeros(M - 1, 1);

while (posicion_ventana + tamanio_ventana < length(audio))
    %Hallo los coeficientes.
    audio_ventaneado = audio(posicion_ventana : (posicion_ventana + tamanio_ventana - 1))
    ;
    r = xcorr(audio_ventaneado);
    r = r(tamanio_ventana : (tamanio_ventana + M - 1));
    R = toeplitz(r(1: M - 1));
    b = R\r(2 : M);
    G = sqrt(r(1) - b' * r(2 : M));

    %Guardo los coeficientes.
    b_matriz(i,:) = b;

    %Guardo Error.
    [error, z] = filter([-1; b], 1, audio_ventaneado(1 : avance_ventana), z);
    error_total = [error_total; error];

    i = i + 1;
    posicion_ventana = posicion_ventana + avance_ventana;
end

%--- Receptor ----

%Debe reconstruir la señal a partir de b_matriz y error_total.
audios = [];

for bits = [16, 8, 6 ,4]

    posicion_ventana = 1;
    i = 1;
    z = zeros(M - 1, 1);
    audio_reconstruido_total = [];

    while (posicion_ventana + tamanio_ventana < length(error_total))
        error_ventaneado = truncar(error_total(posicion_ventana : (posicion_ventana +
    tamanio_ventana - 1)), bits);
        [audio_reconstruido, z] = filter(1, [-1; b_matriz(i,:)'], error_ventaneado(1:
    avance_ventana), z);

        audio_reconstruido_total = [audio_reconstruido_total; audio_reconstruido];

        i = i + 1;
        posicion_ventana = posicion_ventana + avance_ventana;
    end
    audios(bits,:) = audio_reconstruido_total;
end

%Genero Audios
audiowrite('reconstruida_16.wav', audios(16,:), fs);
audiowrite('reconstruida_8.wav',  audios(8, :), fs);
audiowrite('reconstruida_6.wav',  audios(6, :), fs);
audiowrite('reconstruida_4.wav',  audios(4, :), fs);

%Grafico Señales Reconstruidas
figure(1);
hold on;

tiempos = linspace(0, length(audio) / fs, length(audio));
plot(tiempos, audio);
```

```
85 tiempos = linspace(0, length(audios(16,:)) / fs, length(audios(16,:)));

87 plot(tiempos, audios(16,:));
   plot(tiempos, audios(8, :));
89 plot(tiempos, audios(6, :));
   plot(tiempos, audios(4, :));
91
   axis([0.156, 0.174, -0.4, 0.4]);
93 title('Señales Reconstruidas.');
   legend('16 Bits', '8 Bits', '6 Bits', '4 Bits');
95 xlabel('Tiempo (s)');
   ylabel('Amplitud');
97
   %Grafico Errores
99
   figure(2)
101 hold on;

103 audio = audio(1:length(audios(16,:)));
   audio = audio';
105
   plot(tiempos, audio - audios(16,:));
107 plot(tiempos, audio - audios(8,:));
   plot(tiempos, audio - audios(6,:));
109 plot(tiempos, audio - audios(4,:));

111 axis([0.156, 0.174, -0.05, 0.05]);
   title('Errores.');
113 legend('16 Bits', '8 Bits', '6 Bits', '4 Bits');
   xlabel('Tiempo (s)');
115 ylabel('Amplitud');
```

Codigo/ejercicio_2_con_redondeo.m

2.2.2. Señales Reconstruidas

En las figuras 9 y 10 podemos ver la reconstrucción de la señal para diferentes precisiones y sus errores. Puede verse que la magnitud del error al codificar con 4 bits es claramente superior a las demás.

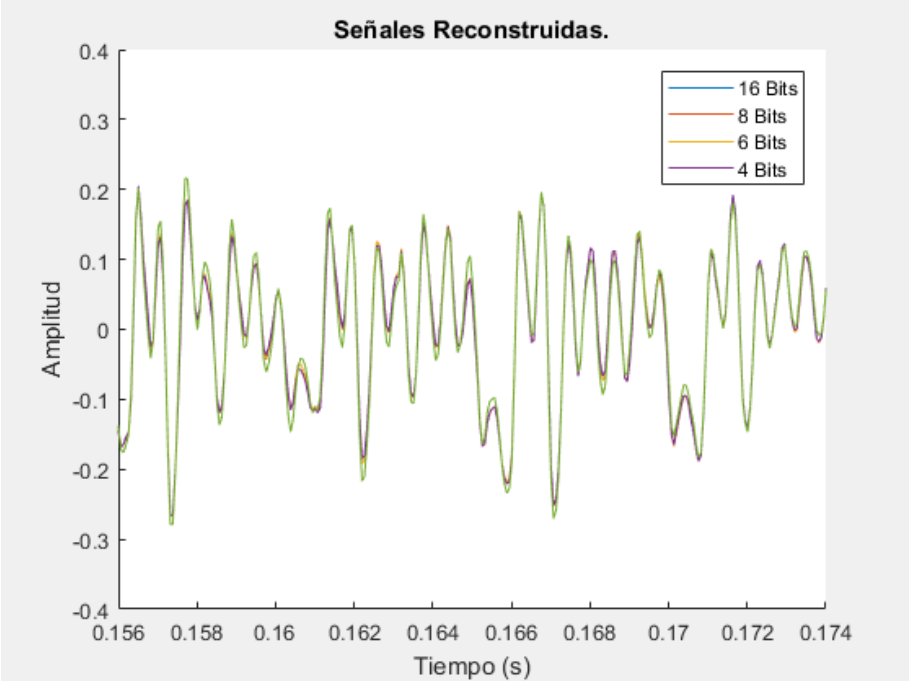

Figura 9: Señales Reconstruidas

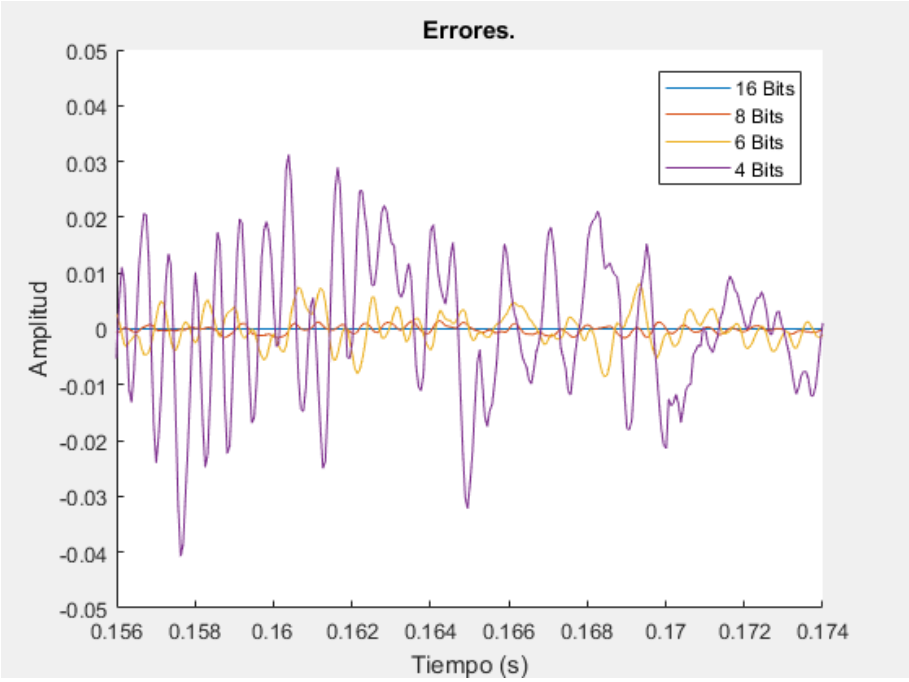

Figura 10: Errores

2.2.3. Memoria

Mediante la instrucción*whos* de Matlab podemos saber que la memoria utilizada para guardar el audio completo es:

$$size(Audio) = 8 * 122889 = 983040bits. \tag{1}$$

Aproximadamente 1 MBit. Mientras que con la codificación LPC:

$$size(AudioLPC) = size(MatrizDeCoeficientesLPC) * 8 + bits * length(Error) \tag{2}$$

Siendo el tamaño de la matriz de coeficientes 15352 Bytes, la longitud del vector de error 15680, y trabajando en 4 bits, podemos llegar a comprimir la señal a unos 185 kBits.

3. Conclusiones

En este trabajo se pudo ver como los coeficientes LPC son útiles tanto para el análisis, como para la codificación y transmisión de señales de voz. Mediante los mismos podemos identificar fonemas mirando la envolvente del espectro. Por otro lado, también se los pudo utilizarlos para codificar una señal de voz utilizando menor cantidad de bits y posteriormente reconstruir satisfactoriamente dicha señal.