6. CÓDIGO FUENTE.

bateria_codificadores.m

```
function bateria_codificadores(fichero,...
                               adpcm01, adpcm02, ...
                               armonico01, armonico02, armonico03, armonico04,
...
                               celp01, celp02, celp03,celp04, celp05, ...
                               dct01, dct02, dct03, ...
                               lpc01, lpc02, lpc03)

% Sintaxis: bateria_codificadores('Original\sonido01.wav',
7,7,20,10,16,6,12,5,7,9,9,0.03,12,6,10,9,5)

% Cierra las figuras y vacia la pantalla
close all
clc

% Lectura de fichero
[voz, frecuencia_muestreo, numero_bits] = wavread(fichero);
voz = voz';

% Frecuencia de trabajo
frecuencia_base = 8000;

% Muestreo a la frecuencia de trabajo
voz = resample(voz, frecuencia_base, frecuencia_muestreo);

% Ajustar la amplitud de la voz a 1
maximo = max([abs(max(voz)) abs(min(voz))]);
voz = voz / maximo;

% Calculo bits datos de entrada
muestras_voz = length(voz);
duracion_fichero = muestras_voz/frecuencia_base;
tamanyo_fichero = muestras_voz*numero_bits;
```

```
tasa_fichero =  tamanyo_fichero/duracion_fichero;

% Analisis/Sintesis ADPCM
    disp('ADPCM')

    % Constantes ADPCM
    muestras_ventana = 160;

    % Asignacion de bits ADPCM
    bits_ganancia = adpcm01;
    bits_error = adpcm02;

    % Ajustar la longitud a un numero entero de ventanas
    numero_ventanas = ceil(muestras_voz/muestras_ventana);
    voz_ADPCM = [voz zeros(1, numero_ventanas*muestras_ventana - muestras_voz)];

    % Tiempo inicio ADPCM
    tic

    % Analisis ADPCM
    [gananciaq, errorq] = analisis_adpcm(voz_ADPCM, bits_ganancia, bits_error);

    % Sintesis ADPCM
    voz_sintetizada = sintesis_adpcm(gananciaq, errorq, bits_ganancia,
bits_error);

    % Tiempo fin ADPCM
    tiempo_ADPCM = toc;

    % Calculo de bits ADPCM
    duracion_voz_ADPCM = length(voz_ADPCM)/frecuencia_base;
    tamanyo_datos_comprimidos = numero_ventanas*(bits_ganancia +
muestras_ventana*bits_error);
    tasa_datos_comprimidos_ADPCM =
tamanyo_datos_comprimidos/duracion_voz_ADPCM;
    ratio_compresion_ADPCM = 100 -
(tamanyo_datos_comprimidos/tamanyo_fichero*duracion_fichero/duracion_voz_ADPCM)
* 100;
```

```
    % Ajuste del rango dinamico y de la media
    voz_sintetizada_ADPCM = ajuste_rango_media(voz_ADPCM, voz_sintetizada);

    % Calculo SNR ADPCM
    SNR_ADPCM = calculo_SNR(voz_ADPCM, voz_sintetizada_ADPCM);

% Analisis/Sintesis Codificador Armonico
    disp('Codificador Armonico')

    % Constantes Codificador Armonico
    muestras_ventana = 160;
    numero_picos = armonico01;

    % Asignacion de bits Codificador Armonico
    bits_amplitudes = armonico02;
    bits_frecuencias = armonico03;
    bits_fases = armonico04;

    % Ajustar la longitud a un numero entero de ventanas
    numero_ventanas = ceil(muestras_voz/muestras_ventana);
    voz_armonico = [voz zeros(1, numero_ventanas*muestras_ventana -
muestras_voz)];

    % Tiempo inicio Codificador Armonico
    tic

    % Analisis Codificador Armonico
    [amplitudesq, frecuenciasq, fasesq, maximo_amplitud] =
analisis_armonico(voz_armonico, numero_picos, bits_amplitudes, bits_frecuencias,
bits_fases);

    % Sintesis Codificador Armonico
    voz_sintetizada = sintesis_armonico(amplitudesq, frecuenciasq, fasesq,
bits_amplitudes, bits_frecuencias, bits_fases);

    % Tiempo fin Codificador Armonico
    tiempo_armonico = toc;

    % Calculo de bits Codificador Armonico
```

```
    duracion_voz_armonico = length(voz_armonico)/frecuencia_base;
    tamanyo_datos_comprimidos = numero_picos*numero_ventanas*(bits_amplitudes +
bits_frecuencias + bits_fases);
    tasa_datos_comprimidos_armonico =
tamanyo_datos_comprimidos/duracion_voz_armonico;
    ratio_compresion_armonico = 100 -
(tamanyo_datos_comprimidos/tamanyo_fichero*duracion_fichero/duracion_voz_armonic
o)*100;

    % Ajuste del rango dinamico y de la media
    voz_sintetizada_armonico = ajuste_rango_media(voz_armonico,
voz_sintetizada);

    % Calculo SNR Codificador Armonico
    SNR_armonico = calculo_SNR(voz_armonico, voz_sintetizada_armonico);

% Analisis/Sintesis CELP
    disp('CELP')

    % Constantes CELP
    muestras_ventana = 240;
    orden_LPC = 10;
    numero_subtramas = 4;

    % Asignacion de bits CELP
    bits_coeficientes = celp01;
    bits_bopt = celp02;
    bits_Topt = celp03;
    bits_indices = celp04;
    bits_ganancia = celp05;

    % Ajustar la longitud a un numero entero de ventanas
    numero_ventanas = ceil(muestras_voz/muestras_ventana);
    voz_CELP = [voz zeros(1, numero_ventanas*muestras_ventana - muestras_voz)];
    numero_tramas = length(voz_CELP)/muestras_ventana;

    % Tiempo inicio CELP
    tic
```

```
    % Analisis CELP
    [coeficientes_lpcq, boptq, Toptq, indicesq, gananciaq] =
analisis_celp(voz_CELP, bits_coeficientes, bits_bopt, bits_Topt, bits_indices,
bits_ganancia);

    % Sintesis CELP
    voz_sintetizada = sintesis_celp(coeficientes_lpcq, boptq, Toptq, indicesq,
gananciaq, bits_coeficientes, bits_bopt, bits_Topt, bits_indices,
bits_ganancia);

    % Tiempo fin CELP
    tiempo_CELP =  toc;

     % Calculo de bits CELP
    duracion_voz_CELP = length(voz_CELP)/frecuencia_base;
    tamanyo_datos_comprimidos = numero_tramas*((orden_LPC
+1)*(bits_coeficientes) + bits_ganancia + numero_subtramas*(bits_bopt +
bits_Topt + bits_indices));
    tasa_datos_comprimidos_CELP =  tamanyo_datos_comprimidos /
duracion_voz_CELP;
    ratio_compresion_CELP = 100 - (tamanyo_datos_comprimidos / tamanyo_fichero *
duracion_fichero / duracion_voz_CELP) * 100;

    % Ajuste del rango dinamico y de la media
    voz_sintetizada_CELP = ajuste_rango_media(voz_CELP, voz_sintetizada);

    % Calculo SNR Codificador Armonico
    SNR_CELP = calculo_SNR(voz_CELP, voz_sintetizada_CELP);

 % Analisis/Sintesis DCT
    disp('DCT')

    % Constantes DCT
    muestras_ventana = 160;
    porcentaje_amplitud = dct01;

    % Asignacion de bits DCT
    bits_indices = dct02;
    bits_coeficientes = dct03;
```

```
    % Ajustar la longitud a un numero entero de ventanas
    numero_ventanas = ceil(muestras_voz/muestras_ventana);
    voz_DCT = [voz zeros(1, numero_ventanas*muestras_ventana - muestras_voz)];

    % Tiempo inicio DCT
    tic

    % Analisis DCT
    [indicesq, coeficientesq, maximo_dct, coeficientes_dct] =
analisis_dct(voz_DCT, porcentaje_amplitud, bits_indices, bits_coeficientes);

    % Sintesis DCT
    voz_sintetizada = sintesis_dct(indicesq, coeficientesq, bits_indices,
bits_coeficientes);

    % Tiempo fin DCT
    tiempo_DCT = toc;

    % Calculo de bits DCT
    duracion_voz_DCT = length(voz_DCT)/frecuencia_base;
    tamanyo_datos_comprimidos = length(indicesq)*bits_indices +
length(coeficientesq)*bits_coeficientes;
    tasa_datos_comprimidos_DCT =  tamanyo_datos_comprimidos/duracion_voz_DCT;
    ratio_compresion_DCT = 100 - (tamanyo_datos_comprimidos/tamanyo_fichero*
duracion_fichero/duracion_voz_DCT)*100;

    % Ajuste del rango dinamico y de la media
    voz_sintetizada_DCT = ajuste_rango_media(voz_DCT, voz_sintetizada);

    % Calculo SNR Codificador Armonico
    SNR_DCT = calculo_SNR(voz_DCT, voz_sintetizada_DCT);

% Analisis/Sintesis LPC
    disp('LPC')

    % Constantes LPC
    muestras_ventana = 240;
```

```
    % Asignacion de bits LPC
    bits_coeficientes = lpc01;
    bits_pitch = lpc02;
    bits_ganancia = lpc03;

    % Ajustar la longitud a un numero entero de ventanas
    numero_ventanas = ceil(muestras_voz/muestras_ventana);
    voz_LPC = [voz zeros(1, numero_ventanas*muestras_ventana - muestras_voz)];

    % Tiempo inicio LPC
    tic

    % Analisis LPC
    [coeficientes_lpcq, pitchq, gananciaq] = analisis_lpc(voz_LPC,
bits_coeficientes, bits_pitch, bits_ganancia);

    % Sintesis LPC
    voz_sintetizada = sintesis_lpc(coeficientes_lpcq, pitchq, gananciaq,
bits_coeficientes, bits_pitch, bits_ganancia);

    % Tiempo fin LPC
    tiempo_LPC = toc;

    % Relleno para igualar la longitud de la fuente y la reconstruccion
    voz_sintetizada = voz_sintetizada';
    voz_sintetizada = [voz_sintetizada zeros(1, length(voz_LPC) -
length(voz_sintetizada))];

     % Calculo de bits LPC
    duracion_voz_LPC = length(voz_LPC)/frecuencia_base;
    tamanyo_datos_comprimidos = numero_ventanas*((orden_LPC+1)*bits_coeficientes
+ bits_pitch + bits_ganancia);
    tasa_datos_comprimidos_LPC =  tamanyo_datos_comprimidos/duracion_voz_LPC;
    ratio_compresion_LPC = 100 -
(tamanyo_datos_comprimidos/tamanyo_fichero*duracion_fichero/duracion_voz_LPC)*10
0;

    % Ajuste del rango dinamico y de la media
    voz_sintetizada_LPC = ajuste_rango_media(voz_LPC, voz_sintetizada);
```

```
    % Calculo SNR LPC
    SNR_LPC = calculo_SNR(voz_LPC, voz_sintetizada_LPC);

%Presentacion de datos
tiempo_minimo = min([tiempo_ADPCM tiempo_armonico tiempo_CELP tiempo_DCT
tiempo_LPC]);
tiempo_ADPCM = round(tiempo_ADPCM/tiempo_minimo);
tiempo_armonico = round(tiempo_armonico/tiempo_minimo);
tiempo_CELP = round(tiempo_CELP/tiempo_minimo);
tiempo_DCT = round(tiempo_DCT/tiempo_minimo);
tiempo_LPC = round(tiempo_LPC/tiempo_minimo);

fprintf('\nPulse una tecla para ver datos\n\n')
pause
close all

fprintf('   Fichero de voz: %s\n',fichero)
fprintf('   Duracion: %.2f s\n',duracion_fichero)
fprintf('   Tasa: %.1d bits/s\n\n',tasa_fichero)
fprintf('   ADPCM %13.2f %.2d %.2f %.2f\n', SNR_ADPCM,
tasa_datos_comprimidos_ADPCM, ratio_compresion_ADPCM, tiempo_ADPCM)
fprintf('   Armonico %10.2f %.2d %.2f %.2f\n', SNR_armonico,
tasa_datos_comprimidos_armonico, ratio_compresion_armonico, tiempo_armonico)
fprintf('   CELP %14.2f  %.1d %.2f %.2f\n', SNR_CELP,
tasa_datos_comprimidos_CELP, ratio_compresion_CELP, tiempo_CELP)
fprintf('   DCT %15.2f %.2d %.2f %.2f\n', SNR_DCT, tasa_datos_comprimidos_DCT,
ratio_compresion_DCT, tiempo_DCT)
fprintf('   LPC %15.2f %.1d %.2f %.2f\n', SNR_LPC, tasa_datos_comprimidos_LPC,
ratio_compresion_LPC, tiempo_LPC)

% Presentacion de graficas
maximo_representacion = max(max([voz voz_sintetizada_ADPCM
voz_sintetizada_armonico voz_sintetizada_CELP voz_sintetizada_DCT
voz_sintetizada_LPC]));
minimo_representacion = min(min([voz voz_sintetizada_ADPCM
voz_sintetizada_armonico voz_sintetizada_CELP voz_sintetizada_DCT
voz_sintetizada_LPC]));
maximo_longitud = max([length(voz) length(voz_sintetizada_ADPCM)
```

```
length(voz_sintetizada_armonico) length(voz_sintetizada_CELP)
length(voz_sintetizada_DCT) length(voz_sintetizada_LPC)]);

fprintf('\nPulse una tecla para ver resultados graficos\n')
pause

subplot(6,1,1), plot(voz)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('Voz Original')

subplot(6,1,2), plot(voz_sintetizada_ADPCM)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('ADPCM')

subplot(6,1,3), plot(voz_sintetizada_armonico)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('Codificador Armonico')

subplot(6,1,4), plot(voz_sintetizada_CELP)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('CELP')

subplot(6,1,5), plot(voz_sintetizada_DCT)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('DCT')

subplot(6,1,6), plot(voz_sintetizada_LPC)
axis([0 maximo_longitud minimo_representacion maximo_representacion])
title('LPC')

% Reproduccion de sonidos
fprintf('\nPulse una tecla reproducir la voz original\n\n')
pause
sound(voz)

fprintf('Pulse una tecla reproducir la voz procedente del Codificador
ADPCM\n\n')
pause
sound(voz_sintetizada_ADPCM)
```

```
fprintf('Pulse una tecla reproducir la voz procedente del Codificador
Armonico\n\n')
pause
sound(voz_sintetizada_armonico)

fprintf('Pulse una tecla reproducir la voz procedente del Codificador CELP\n\n')
pause
sound(voz_sintetizada_CELP)

fprintf('Pulse una tecla reproducir la voz procedente del Codificador DCT\n\n')
pause
sound(voz_sintetizada_DCT)

fprintf('Pulse una tecla reproducir la voz procedente del Codificador LPC\n\n')
pause
sound(voz_sintetizada_LPC)

% Grabacion de ficheros
maximo_absoluto = max(abs([maximo_representacion minimo_representacion]));
wavwrite(voz/maximo_absoluto, 8000, 8, 'Sintetizado/original.wav')
wavwrite(voz_sintetizada_ADPCM/maximo_absoluto, 8000, 8,
'Sintetizado/adpcm.wav')
wavwrite(voz_sintetizada_armonico/maximo_absoluto, 8000, 8,
'Sintetizado/armonico.wav')
wavwrite(voz_sintetizada_CELP/maximo_absoluto, 8000, 8, 'Sintetizado/celp.wav')
wavwrite(voz_sintetizada_DCT/maximo_absoluto, 8000, 8, 'Sintetizado/dct.wav')
wavwrite(voz_sintetizada_LPC/maximo_absoluto, 8000, 8, 'Sintetizado/lpc.wav')

% Cierra las figuras y vacia la pantalla
fprintf('Pulse una tecla para terminar\n\n')
pause
clc

function voz_sintetizada = ajuste_rango_media(voz, voz_sintetizada)

% Ajuste de energia
energia = sqrt(sum(voz.*voz));
energia_sintetizada = sqrt(sum(voz_sintetizada.*voz_sintetizada));
```

```
voz_sintetizada = voz_sintetizada * energia / energia_sintetizada;

% Ajuste de media
media_voz = sum(voz)/length(voz);
media_sintetizada = sum(voz_sintetizada)/length(voz_sintetizada);
voz_sintetizada = voz_sintetizada + media_voz - media_sintetizada;

function SNR = calculo_SNR(voz, voz_sintetizada)

% Calculo SNR
diferencia = voz - voz_sintetizada;
N = sum(diferencia.^2);
S = sum(voz_sintetizada.^2);
SNR = 10*log(S/N);
```

analisis_adpcm.m

```
function [gananciaq, errorq] = analisis_adpcm(voz, bits_ganancia, bits_error)
% gananciaq: ganacia cuantizada con 'bits_ganancia' bits
% errorq: error cuantizado con 'bits_error' bits

% Analisis

% Calculo del numero de ventanas
muestras_ventana = 160;
muestras_voz = length(voz);
numero_ventanas = muestras_voz / muestras_ventana;

% Calculo de la ganancia para cada ventana
for k = 1:numero_ventanas
    ganancia(k) = max(voz(1+(k-1)*muestras_ventana:k*(muestras_ventana))) +
0.001;
end

% Cuantizacion de la ganancia
gananciaq = uencode(ganancia, bits_ganancia);

% Calculo del error diferencial en cada ventana
```

```
for k = 1:numero_ventanas
    % Normalizacion por la ganancia
    voz_normalizada  = voz(1+(k-1)*muestras_ventana:k*(muestras_ventana)) /
ganancia(k);

    % xp[0] = 0
    muestra_anterior(1) = 0;

    % Calculo del error usando predictor de orden uno
    for l = 1:muestras_ventana
        % e[n] = x[n] - xp[n]
        error(k,l) = voz_normalizada(l) - muestra_anterior(l);
        maximo = max(abs([max(error(k,:)) min(error(k,:))]));

        if maximo == 0
            maximo = 1;
        end

        % Cuantizacion del error dentro de la ventana
        error_temporalq = uencode(error(k,:), bits_error, maximo);

        % Decuantizacion del error dentro de la ventana
        error_temporaldq = udecode(error_temporalq, bits_error, maximo);

        % xp[n] = xp[n-1] + edq[n-1]
        muestra_anterior(l+1) = muestra_anterior(l) + error_temporaldq(l);
    end

    % Cuantizacion del error de todas las ventanas
    maximo_error(k) = max(abs([max(error(k,:)) min(error(k,:))]));
    errorq(k,:) = uencode(error(k,:), bits_error,maximo_error(k));
end
```

sintesis_adpcm.m

```
function voz_sintetizada = sintesis_adpcm(gananciaq, errorq, bits_ganancia,
bits_error)
% voz_sintetizada: voz reconstruida a partir de los parametros del analisis
```

```
% Sintesis

% Constantes
muestras_ventana = 160;

% Calculo del numero de ventanas
numero_ventanas = length(gananciaq);

% Decuantizacion de la ganancia
ganancia = udecode(gananciaq, bits_ganancia);

% Decuantizacion del error
for k = 1: numero_ventanas
    error(k,:) = udecode(errorq(k,:), bits_error);
end

% Reconstruccion de la señal a partir del error diferencial
for k = 1:numero_ventanas
    % xp[0] = 0
    muestra_anterior(1) = 0;

    for l = 1:muestras_ventana
        % x[n] = e[n] + xp[n]
        voz_normalizada(l) = error(k,l) + muestra_anterior(l);

        % xp[n] = x[n-1]
        muestra_anterior(l+1) = voz_normalizada(l);
    end

    % Escalado de la voz_normalizada
    voz_sintetizada(1+(k-1)*muestras_ventana:k*(muestras_ventana)) =
voz_normalizada * ganancia(k);
end
```

analisis_armonico.m

```
function [amplitudesq, frecuenciasq, fasesq, maximo_amplitud] =
```

```
analisis_armonico(voz, numero_picos, bits_amplitudes, bits_frecuencias,
bits_fases)
% amplitudesq: amplitudes cuantizadas con 'bits_amplitudes' bits
% frecuenciasq: frecuencias cuantizadas con 'bits_frecuencias' bits
% fasesq: fases cuantizadas con 'bits_fases' bits
% maximo_amplitud: maximo en valor absoluto devuelto para evaluar la compresion

% Constantes
frecuencia_base = 8000;
muestras_ventana = 160;
puntos_fft = 1024;

% Analisis
muestras_voz = length(voz);
numero_ventanas = muestras_voz / muestras_ventana;

% Calculo de la DFT y seleccion de parametros
for k = 1:numero_ventanas
    voz_segmentada = voz((muestras_ventana*(k-1)+1):(muestras_ventana*k));
    transformada_fft = fft(voz_segmentada,puntos_fft);
    magnitud_fft = abs(transformada_fft(1:puntos_fft/2));
    %[amplitudes_fft, argumentos_fft] = busca_picos(magnitud_fft, numero_picos);
    [amplitudes_fft, argumentos_fft] = busca_ventanas(magnitud_fft,
numero_picos);

    amplitudes(:,k) = amplitudes_fft;
    frecuencias(:,k) = argumentos_fft*(2*pi/puntos_fft);
    fases(:,k) = angle(transformada_fft(argumentos_fft));
end

% Calculo de los maximos para la cuantizacion
maximo_amplitud = max(max(amplitudes));
maximo_frecuencia = puntos_fft/2;
maximo_fase = 2*pi;

% Cuantizacion
amplitudesq = uencode(amplitudes,bits_amplitudes, maximo_amplitud);
frecuenciasq = uencode(frecuencias,bits_frecuencias, maximo_frecuencia);
fasesq = uencode(fases,bits_fases, maximo_fase);
```

```
function [amplitudes_fft, argumentos_fft] = busca_picos(magnitud_fft,
numero_picos)
% Devuelve los 'numero_picos' pares {amplitud, argumento} maximos de
% magnitud_fft

[magnitud_ordenada, indices] = sort(magnitud_fft,2,'descend');
amplitudes_fft = magnitud_ordenada(1:numero_picos);
argumentos_fft = indices(1:numero_picos);

function [amplitudes_fft, argumentos_fft] = busca_ventanas(magnitud_fft,
numero_picos)
% Devuelve los 'numero_picos' pares {amplitud, argumento} de los puntos mas
% notables de aquellas ventanas  en las que se producen las diferencias mas
% significativas de magnitud_fft

% Las diferencias se estableceran entre ventanas correlativas de 4 muestras
muestras_v = 4;
numero_v = 1024/8;

% Calculo de la energia por ventana
for k = 1 : numero_v
   energia_v(k) = sum(magnitud_fft((k-1)*4+1:k*4).^2);
end

% Calculo de la diferencia de energia entre ventanas
for l = 1 : numero_v - 1
    diferencia_v(l) = energia_v(l+1) - energia_v(l);
end

% Ordenar las energias de mayor a menor
[diferencia_ordenada,indices]=sort(diferencia_v,'descend');

% Obtencion de la amplitud y el argumento del punto mas significativo de la
% ventana previa
for m = 1 : numero_picos
   [amplitud, posicion_v] = max(magnitud_fft((indices(m)-1)*muestras_v +
1:indices(m)*muestras_v));
   amplitudes_fft(m) = amplitud;
```

```
   argumentos_fft(m) = posicion_v + (indices(m)-1)*muestras_v;
end
```

sintesis_armonico.m

```
function voz_sintetizada = sintesis_armonico(amplitudesq, frecuenciasq, fasesq,
bits_amplitudes, bits_frecuencias, bits_fases)
% voz_sintetizada: voz reconstruida a partir de los parametros del analisis.

% Constantes
frecuencia_base = 8000;
muestras_ventana = 160;
puntos_fft = 1024;

% Decuantizacion
maximo_frecuencia = puntos_fft/2;
maximo_fase = 2*pi;

amplitudes = udecode(amplitudesq, bits_amplitudes);
frecuencias = udecode(frecuenciasq, bits_frecuencias, maximo_frecuencia);
fases = udecode(fasesq, bits_fases, maximo_fase);

% Sintesis

[numero_picos, numero_ventanas] = size(amplitudesq);

% Reconstruccion de la señal como suma de senoides
for k = 1:numero_ventanas
    voz_sintetizada(muestras_ventana*(k-1)+1:muestras_ventana*k) =
1/puntos_fft*sum((amplitudes(:,k)*ones(1,muestras_ventana)).*cos(fases(:,k)*ones
(1,muestras_ventana) + frecuencias(:,k)*(1:muestras_ventana)));
end
```

analisis_dct.m

```
function [indicesq, coeficientesq, maximo_dct, coeficientes_dct] =
```

```
analisis_dct(voz, porcentaje_amplitud, bits_indices, bits_coeficientes)
% indicesq: indices cuantizados con 'bits_indices' bits
% coeficientesq: coeficientes cuantizados con 'bits_coeficientes' bits
% maximo_dct: maximo en valor absoluto devuelto para evaluar la compresion
% coeficientes_dct: numero de coeficientes que superan el umbral para evaluar la
compresion

% Constantes
frecuencia_base = 8000;
muestras_ventana = 20 * frecuencia_base /1000;

% Analisis

muestras_voz = length(voz);
numero_ventanas = muestras_voz / muestras_ventana;

% Enventanado de la voz y obtencion de coeficientes DCT
for i = 1:numero_ventanas
    voz_segmentada(i,1:muestras_ventana)= voz((muestras_ventana*(i-1)+1):
(muestras_ventana*i));
    coeficientes_dct(i,1:muestras_ventana) = dct(voz_segmentada(i,
1:muestras_ventana));
end

% Mayor coeficiente DCT que servira de unmbral
maximo = max(max(coeficientes_dct));
minimo = min(min(coeficientes_dct));
maximo_dct = max([abs(maximo) abs(minimo)]);

indices = [];
indices(1) = numero_ventanas;
indices(2) = muestras_ventana;

coeficientes = [];

for i = 1:numero_ventanas
    for j = 1:muestras_ventana
        if abs(coeficientes_dct(i,j)) >= maximo_dct * porcentaje_amplitud
            indices(length(indices) + 1) = j;
```

```
            coeficientes(length(coeficientes) + 1) = coeficientes_dct(i,j);
        end
    end

    indices(length(indices) + 1) = 0;
    coeficientes(length(coeficientes) + 1) = 0;
end

% Cuantizacion
indicesq = uencode(indices,bits_indices,max(muestras_ventana,numero_ventanas));
coeficientesq = uencode(coeficientes,bits_coeficientes, maximo_dct);
```

sintesis_dct.m

```
function voz_sintetizada = sintesis_dct(indicesq, coeficientesq, bits_indices,
bits_coeficientes)
% voz_sintetizada: voz reconstruida a partir de los parametros del analisis

% Constantes
frecuencia_base = 8000;
muestras_ventana = 20 * frecuencia_base /1000;

% Decuantizacion
indices = ceil(udecode(indicesq,bits_indices,muestras_ventana));
[uno,columnas] = size(indices);

% Recuperacion del numero de ventanas
m = 1;

for k = 3 : columnas
    if indices(k) == 0
        m = m + 1;
    end
end

numero_ventanas = m - 1;

% Recuperacion de la ganancia
```

```
ganancia =  max(numero_ventanas,muestras_ventana);

% Recuperacion de los indices
indices = round(udecode(indicesq,bits_indices,ganancia));
coeficientes = udecode(coeficientesq,bits_coeficientes);

% Recuperacion de los coeficientes
coeficientes_dct = zeros(numero_ventanas,muestras_ventana);
m = 1;
for k = 3 : columnas
    if indices(k) ~= 0
        coeficientes_dct(m,indices(k)) = coeficientes(k-2);
    else
        m = m + 1;
    end
end

% Sintesis

% Reconstruccion usando la transformada inversa de los coeficientes
for i = 1:(m-1)
    voz_sintetizada((muestras_ventana*(i-1)+1):(muestras_ventana*i)) =
idct(coeficientes_dct(i,1:muestras_ventana));
end
```

analisis_celp.m

```
function [coeficientes_lpcq, boptq, Toptq, indicesq, gananciaq] =
analisis_CELP(voz, bits_coeficientes, bits_bopt, bits_Topt, bits_indices,
bits_ganancia)
% coeficientes_lpc: matriz de coeficientes LPC cuantizados
% boptq: matriz de ganancias LTP
% Toptq: matriz de periodos de pitchh
% indicesq: matriz de indices del libro de codigos
% gananciaq: matriz de ganancias STP

% Constantes
muestras_trama = 240;
```

```
muestras_subtrama = 60;
numero_subtramas = 4;

% Orden del filtro LPC correspondiente al analisis de prediccion corto
orden_LPC = 10;

% Periodos del barrido de pitch
Tmin = 20;
Tmax = 147;

% Buffer de señal de entrada para el analisis 'long-term'
buffer_old = zeros(1,Tmax - Tmin + muestras_subtrama);

% Banderas y estado de trama
TRUE = 1;
FALSE = 0;
silencio = FALSE;

% Pametro gamma de filtrado
gamma = 0.9;
gamma2 = 0.5;

% Libro aleatorio
numero_codigos = 128;
fid = fopen('codebook_ternario.bin','rb');
libro_codigos = fread(fid, [muestras_subtrama, numero_codigos],'float64');
fclose(fid);

% Analisis
numero_tramas = length(voz)/muestras_trama;
for indice_trama = 1 : numero_tramas
    % Enventanado
    trama = voz(muestras_trama*(indice_trama - 1) + 1 :
muestras_trama*indice_trama);

    % Analisis 'short-term'
    coeficientes_trama(indice_trama, :) = lpc(trama, orden_LPC);

    % Promediado de coeficientes LPC
```

```
    coeficientes_trama_promediados(1) = coeficientes_trama(indice_trama,1);
    coeficientes_trama_promediados2(1) = coeficientes_trama(indice_trama,1);

    for indice = 2 : orden_LPC+1
        coeficientes_trama_promediados(indice) =
coeficientes_trama(indice_trama, indice)*(gamma^(indice-1));
        coeficientes_trama_promediados2(indice) =
coeficientes_trama(indice_trama,indice)*(gamma2^(indice-1));
    end

    % Filtrado de error de prediccion para el analisis 'long-term¡
    trama_predicha = filter(-1*coeficientes_trama(indice_trama, 2 : orden_LPC
+1), 1, trama);
    error_trama = trama - trama_predicha;

   % Analisis 'long-term'
    for indice_subtrama = 1 : numero_subtramas
        % Enventanado
        subtrama = trama(muestras_subtrama*(indice_subtrama-1) + 1 :
muestras_subtrama*indice_subtrama);

        error_subtrama = error_trama(muestras_subtrama*(indice_subtrama-1) + 1 :
muestras_subtrama*indice_subtrama);

        buffer_old = [buffer_old(muestras_subtrama - Tmin + 1 : Tmax - Tmin +
muestras_subtrama) error_subtrama(1 : muestras_subtrama - Tmin)];

        silencio = TRUE;
        Jmin = inf;

        % Barrido de pitch
        for T = Tmin : Tmax
            % Caso de que para algun pitch's la energia del buffer no sea
            % cero
            if(sum(buffer_old(Tmax - T + 1: Tmax - T +
muestras_subtrama).*buffer_old(Tmax - T + 1: Tmax - T + muestras_subtrama))) ~=
0
                silencio = FALSE;
```

```
                b = -sum(error_subtrama.*buffer_old(Tmax - T + 1: Tmax - T +
muestras_subtrama))/sum(buffer_old(Tmax - T + 1: Tmax - T +
muestras_subtrama).*buffer_old(Tmax - T + 1: Tmax - T + muestras_subtrama));
                J = sum((error_subtrama-buffer_old(Tmax - T + 1: Tmax - T +
muestras_subtrama)).*(error_subtrama-buffer_old(Tmax - T + 1: Tmax - T +
muestras_subtrama)));

                % Seleccion de las combinaciones de ganancia y de pitch
                % optimas, aquellas que minimizan la distorsion
                if(J < Jmin)
                    Jmin = J;
                    bopt = b;
                    Topt = T;
                end
            end

        end

        if silencio == TRUE
            btabla(indice_trama, indice_subtrama) = 0;
        else
            btabla(indice_trama, indice_subtrama) = bopt;
            Ttabla(indice_trama, indice_subtrama) = Topt;
        end

        % Filtrado perceptual de la subtrama
        subtrama_promediada = filter(coeficientes_trama_promediados2,
coeficientes_trama_promediados, subtrama);

        % Calculo de la energia de la subtrama
        energia_subtrama_promediada =
sqrt(sum(subtrama_promediada.*subtrama_promediada));

        % Calculo de la ganancia
        R = xcorr(trama);
        Rs = R(muestras_trama + [0 : orden_LPC]);

        J(1) = Rs(1);
        k(1) = 0;
```

```
        A = [];

        for l = 2 : orden_LPC + 1
            numerador_ld = [1 A']*Rs(l:-1:2)';
            denominador_ld = -J(l-1);
            k(l) = numerador_ld/denominador_ld;
            A = [A + k(l)*flipud(A); k(l)];
            J(l) = (1-k(l)^2)*J(l-1);
        end

        ganancia_ld = sqrt(J(orden_LPC + 1));

        % Barrido del libro de codigos
        EMSmin = inf;

        for h = 1 : numero_codigos
           % Filtrado de sintesis de pitch
            codigo = libro_codigos(:,h)';

            if btabla(indice_trama, indice_subtrama) ~= 0
                denominador = zeros(1, Ttabla(indice_trama, indice_subtrama)+1);
                denominador(1) = 1;
                denominador(Ttabla(indice_trama, indice_subtrama)+1) =
btabla(indice_trama, indice_subtrama);

                salida_filtro_pitch = filter(1, denominador, codigo);
            else
                salida_filtro_pitch = codigo;
            end

            % Filtrado de sintesis de formantes
            subtrama_reconstruida = filter(1, coeficientes_trama(indice_trama),
salida_filtro_pitch);
            subtrama_reconstruida = filter(coeficientes_trama_promediados2,
coeficientes_trama_promediados, subtrama_reconstruida);

            % Escalado por la ganancia
            subtrama_reconstruida = ganancia_ld * subtrama_reconstruida;
```

```
            % Calculo del error cuadratico medio
            diferencia = subtrama_promediada - subtrama_reconstruida;
            EMS = sum(diferencia.*diferencia);

            if(EMS < EMSmin)
                EMSmin = EMS;
                indice_libro(indice_trama, indice_subtrama) = h;
                sr(indice_trama, indice_subtrama, :) = subtrama_reconstruida;
                ganancia(indice_trama, indice_subtrama) = ganancia_ld;
            end
        end
    end
end

% Maximos para cuantizacion
coeficiente_lpc_maximo = 2;
bmaximo = 21;
Tmaximo =  147;
indice_maximo = numero_codigos;
ganancia_maxima = 6;

% Cuantizacion
coeficientes_lpcq = uencode(coeficientes_trama, bits_coeficientes,
coeficiente_lpc_maximo);
boptq = uencode(btabla, bits_bopt, bmaximo);
Toptq = uencode(Ttabla, bits_Topt, Tmaximo);
indicesq = uencode(indice_libro, bits_indices, indice_maximo);
gananciaq = uencode(ganancia, bits_ganancia, ganancia_maxima);
```

sintesis_celp.m

```
function voz_sintetizada = sintesis_CELP(coeficientes_lpcq, boptq, Toptq,
indicesq, gananciaq, bits_coeficientes, bits_bopt, bits_Topt, bits_indices,
bits_ganancia)
% voz_sintetizada: voz reconstruida a partir de los parametros del analisis

% Constantes
```

```
muestras_trama = 240;
muestras_subtrama = 60;
numero_subtramas = 4;

% Orden del filtro LPC correspondiente al analisis de prediccion corto
orden_LPC = 10;

% Periodos del barrido de pitch
Tmin = 20;
Tmax = 147;

% Buffer de señal de entrada para el analisis 'long-term'
buffer_old = zeros(1,Tmax - Tmin + muestras_subtrama);

% Pametro gamma de filtrado
gamma = 0.9;

% Libro aleatorio
numero_codigos = 128;
fid = fopen('codebook_ternario.bin','rb');
libro_codigos = fread(fid, [muestras_subtrama, numero_codigos],'float64');
fclose(fid);

% Numero de tramas
[numero_tramas vacio] = size(coeficientes_lpcq);

% Maximos para decuantizacion
coeficiente_lpc_maximo = 2;
bmaximo = 21;
Tmaximo =  147;
indice_maximo = numero_codigos;
ganancia_maxima = 6;

% Decuantizacion
coeficientes_lpcdq = udecode(coeficientes_lpcq, bits_coeficientes,
coeficiente_lpc_maximo);
boptdq = udecode(boptq, bits_bopt, bmaximo);
Toptdq = round(udecode(Toptq, bits_Topt, Tmaximo));
indicesdq = round(udecode(indicesq, bits_indices, indice_maximo));
```

```
gananciadq = udecode(gananciaq, bits_ganancia, ganancia_maxima);

% Sintesis

for indice_trama = 1 : numero_tramas
    for indice_subtrama = 1 : numero_subtramas
        % Promediado de coeficientes LPC
        coeficientes_trama_promediados(1) = coeficientes_lpcdq(indice_trama,1);
        for indice = 2 : orden_LPC+1
            coeficientes_trama_promediados(indice) =
coeficientes_lpcdq(indice_trama, indice)*(gamma^(indice-1));
        end

        % Seleccion de la secuencia de excitacion
        codigo = libro_codigos(:,indicesdq(indice_trama, indice_subtrama));

        % Escalado por la ganancia
        entrada_filtro_pitch = gananciadq(indice_trama, indice_subtrama)*codigo;

        % Paso por el filtro de sintesis de pitch
        if boptdq(indice_trama, indice_subtrama) ~= 0
            denominador = zeros(1,Toptdq(indice_trama, indice_subtrama)+1);
            denominador(1) = 1;
            denominador(Toptdq(indice_trama, indice_subtrama)) =
boptdq(indice_trama, indice_subtrama);

            salida_filtro_pitch = filter(1, denominador, entrada_filtro_pitch);
        else
            salida_filtro_pitch = entrada_filtro_pitch;
        end

        % Paso por el filtro de sintesis de formantes
        denominador = coeficientes_trama_promediados;
        subtrama_reconstruida = (filter(1, denominador,salida_filtro_pitch))';

        voz_sintetizada((indice_trama-1)*muestras_trama +
(indice_subtrama-1)*muestras_subtrama + 1 : (indice_trama-1)*muestras_trama +
indice_subtrama*muestras_subtrama) = subtrama_reconstruida;
    end
```

```
end
```

codebook_aleatorio.m

```
function libro = codebook_aleatorio(numero_muestras, numero_elementos)
% libro: libro de codigos aleatorio

libro = randn(numero_muestras, numero_elementos);
energia_libro = sqrt(sum(libro.*libro));

for k = 1: numero_elementos
    for l = 1: numero_muestras
        libro(l,k) = libro(l,k)/energia_libro(k);
    end
end

fid = fopen('codebook_aleatorio.bin','wb');
fwrite(fid,libro,'float64');
fclose(fid);
```

codebook_ternario.m

```
function libro = codebook_ternario(numero_muestras, numero_elementos)
% libro: libro de codigos ternario

libro_temporal = rand(numero_muestras, numero_elementos);

for l = 1: numero_elementos
    for k = 1: numero_muestras
        if libro_temporal(k,l) > 2/3
            libro(k,l) = 1;
        elseif libro_temporal(k,l) < 1/3
            libro(k,l) = -1;
        else
            libro(k,l) = 0;
        end
    end
```

```
end

energia_libro = sqrt(sum(libro.*libro));

for k = 1: numero_elementos
    for l = 1: numero_muestras
        libro(l,k) = libro(l,k)/energia_libro(k);
    end
end

fid = fopen('codebook_ternario.bin','wb');
fwrite(fid,libro,'float64');
fclose(fid);
```

analisis_lpc.m

```
function [coeficientes_lpcq, pitchq, gananciaq] = analisis_LPC(voz,
bits_coeficientes, bits_pitch, bits_ganancia)
% coeficientes_lpcq: coeficientes_lpc cuantizados con 'bits_coeficientes' bits
% pitchq: pitch cuantizado con 'bits_pitch' bits
% ganancia; ganacia cuantizada con 'bits_ganancia' bits

% Analisis

% Numero de muestras
muestras_voz = length(voz);

% Parametros LPC
orden_lpc = 10;
muestras_analisis = 160;
muestras_trama = 240;

Pmax = 147;
Pmin = 20;

indice_trama = 0;
numero_tramas = floor((muestras_voz-muestras_analisis)/muestras_analisis);
```

```
for indice_trama = 1:numero_tramas
    % Se analizan tramas de 30 milisegundos que se solapan 10 milisegundos
    % con la anterior
    segmento_analisis = voz((indice_trama-1)*muestras_analisis+1 :
(indice_trama-1)*muestras_analisis + muestras_trama);

    % Inicializadon del buffer para el calculo de la autocorrelacion
    if indice_trama == 1
        buffer = zeros(muestras_trama + Pmax - Pmin,1);
    else
        buffer = [buffer(muestras_trama - Pmax + 1 : muestras_trama);
segmento_analisis(1 : muestras_trama - Pmin)'];
    end

    % Calculo de los coeficientes LPC mediante el metodo de Levinson-Durbin
    R = xcorr(segmento_analisis);
    Rs = R(muestras_trama + [0 : orden_lpc]);
    J(1) = Rs(1);
    k(1) = 0;
    A = [];
    for l = 2 : orden_lpc + 1
        numerador = [1 A']*Rs(l:-1:2)';
        denominador = -J(l-1);
        k(l) = numerador/denominador;
        A = [A + k(l)*flipud(A); k(l)];
        J(l) = (1-k(l)^2)*J(l-1);
    end
    coeficientes_lpc(:,indice_trama) = [1 ; A];

    % Calculo de la ganancia
    ganancia(indice_trama) = sqrt(J(orden_lpc + 1));

    % Filtrado del segmento de analisis y obtencion del residuo para el
    % calculo del pitch
    errorc = filter([1 A'],1,segmento_analisis);
    pico(indice_trama) = 0;
    pitch(indice_trama) = 0;
```

```
    for m = Pmin : Pmax
        autocorrelacion = 0;
        for n = 1:muestras_trama
            autocorrelacion = autocorrelacion + segmento_analisis(n)*buffer(n-m
+Pmax);
        end
        a(m) = autocorrelacion;
        if autocorrelacion > pico(indice_trama)
            pico(indice_trama) = autocorrelacion;
            pitch(indice_trama) = m;
        end
    end
end


% Maximos para la cuantizacion
coeficiente_lpc_maximo = 2;
pitch_maximo = 147;
ganancia_maxima = 6;

% Cuantizacion
coeficientes_lpcq = uencode(coeficientes_lpc, bits_coeficientes,
coeficiente_lpc_maximo);
pitchq =  uencode(pitch, bits_pitch, pitch_maximo);
gananciaq = uencode(ganancia, bits_ganancia, ganancia_maxima);
```

sintesis_lpc.m

```
function voz_sintetizada = sintesis_LPC(coeficientes_lpcq, pitchq, gananciaq,
bits_coeficientes, bits_pitch, bits_ganancia)
% voz_sintetizada: voz reconstruida a partir de los parametros del analisis.

% Parametros LPC
muestras_analisis = 160;
muestras_trama = 240;
muestras_solapadas = 80;

% Maximos para la decuantizacion
```

```
coeficiente_lpc_maximo = 2;
pitch_maximo = 147;
ganancia_maxima = 6;

% Decuantizacion
coeficientes_lpcdq = udecode(coeficientes_lpcq, bits_coeficientes,
coeficiente_lpc_maximo);
pitchdq =  round(udecode(pitchq, bits_pitch, pitch_maximo));
gananciadq = udecode(gananciaq, bits_ganancia, ganancia_maxima);

% Sintesis

[orden_lpc numero_tramas] = size(pitchq);
voz_sintetizada = [];

for indice_trama = 1 : numero_tramas
    % Vector de coeficientes LPC para una trama dada
    A = coeficientes_lpcdq(:, indice_trama);

    if pitchdq(indice_trama) >= 40
        % Segmento sonoro, se usa como señal base un tren de impulsos
        % periodicos
        senyal_base =
normalizador(generador(pitchdq(indice_trama),muestras_trama));
    else
        % Segmento sordo, se usa como señal base una secuencia aleatoria
        senyal_base = normalizador(randn(muestras_trama,1));
    end

    % Se filtra la señal base, sintetizando el segmento de voz
    segmento_sintesis = filter(gananciadq(indice_trama), A', senyal_base);

    % Se solapan las tramas en un segmento de 80 muestras
    if indice_trama == 1
        voz_sintetizada = [voz_sintetizada; segmento_sintesis];
    else
        voz_sintetizada = [voz_sintetizada(1 : length(voz_sintetizada) -
muestras_solapadas); 0.5*voz_sintetizada(length(voz_sintetizada) -
muestras_solapadas +1 : length(voz_sintetizada)) + 0.5*segmento_sintesis(1 :
```

```
muestras_solapadas) ; segmento_sintesis(1 + muestras_solapadas :
muestras_trama)];
    end
end

function tren = generador(periodo,total)
% Devuelve un tren de impulsos periodicos de periodo 'periodo' y longitud
%   'total'
tren = zeros(total,1);
tren(1 : periodo:total) = 1;

function senyal = normalizador(senyal)
% Devuelve la secuencia de entrada normalizada
energia = sum(senyal.^2);
senyal = senyal/sqrt(energia);
```