

MEMORIA DE PROGRAMACIÓN

**Interfaz Gráfico de Análisis y Procesado de Señales de Sonido
en Matlab: App Designer**

Fsi005

Arturo Labajo

Rubén Urraca

ÍNDICE

1-	Introducción	3
2-	Funciones	4
1-	startupFcn	4
2-	Cargar_Archivo.....	4
3-	UpdatePlot.....	6
4-	Grabar	6
5-	Parar_Grabación	7
6-	Reproducir	8
7-	Pausa.....	9
8-	Stop.....	9
9-	Señal_Generada.....	10
10-	Invertir_Senal	12
11-	Histograma	13
12-	CompresionExpansion	14
13-	Efectos	14
14-	Ruido	16
15-	CambiarPanel	18

1- Introducción

El objetivo de esta memoria consta de documentar nuestro desarrollo para la realización de una interfaz de sonido creada a través de MATLAB utilizando AppDesigner.

La finalidad del proyecto es proporcionar una interfaz gráfica interactiva diseñada para ser intuitiva y accesible con fácil manejo (ver Figura 1).

Esta interfaz permitirá al usuario además de cargar archivos de audio y reproducirlos, pueden ver el espectrograma y el histograma correspondiente, añadir efectos, ruidos y muchas más cosas que iremos viendo tanto en el manual de usuario como adelante en este manual de programación.

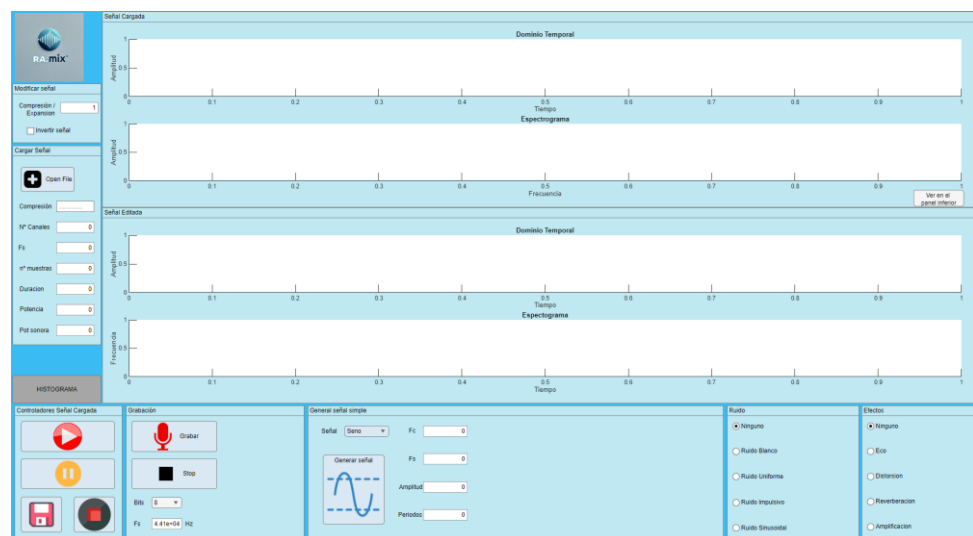


Figura 1: Interfaz

Para realizar esto hemos añadido objetos para poder interactuar y pulsando encima de ellos con el clic derecho y añadiendo un “callback” hemos podido hacer que cada objeto con el que el usuario pueda interactuar tenga una funcionalidad distinta al resto de manera muy sencilla

2- Funciones

1- startupFcn

Esta función se ejecuta al iniciar la aplicación y lo que queremos es que nos maximice la ventana para que se ajuste al tamaño de la pantalla que reproduce la aplicación (ver Figura 2)

```
% Code that executes after component creation
function startupFcn(app)
    app.UIFigure.WindowState = 'maximized';
end
```

Figura 2: Función startupFcn

2- Cargar_Archivo

Con esta función buscamos que el usuario cargue un archivo de audio desde su dispositivo.

En la primera parte de esta función (ver Figura 3), lo que realizamos es la selección del archivo mediante **uigetfile** obteniendo el nombre y camino del archivo, tras esto comprobamos que se ha seleccionado correctamente el archivo y se procede a cargarlo mediante **audioread**, almacenando la pista de audio y la frecuencia de muestreo.

```
% Button pushed function: OpenFileButton
function Cargar_Archivo(app, event)
    % Seleccionar archivo de audio
    [filename, filepath] = uigetfile({'*.wav;*.mp3', 'Audios (*.wav, *.mp3)'});

    % Verificar si el usuario seleccionó un archivo
    if isequal(filename, 0)
        uialert(app.UIFigure, 'No se seleccionó ningún archivo', 'Error');
        return; % Salir si no se seleccionó ningún archivo
    end

    % Cargar archivo de audio
    fullFileName = fullfile(filepath, filename);
    [app.audioData, app.fs] = audioread(fullFileName); % Cargar los datos de audio y la frecuencia de muestreo

    % Si ya existe un reproductor, detenerlo antes de actualizar
    if exist('audioPlayer', 'var') && ~isempty(app.player) && isvalid(app.player)
        stop(app.player);
    end
    app.audioData = mean(app.audioData, 2);
    % Crear un nuevo objeto de reproducción
    app.player = audioplayer(app.audioData, app.fs);
```

Figura 3: Primera parte función Cargar_Archivo

En la segunda parte de esta función (ver Figura 4), lo que hacemos es obtener los datos interesantes del archivo para mostrárselo al usuario en la interfaz (ver Figura 5). Algunos datos los obtenemos a través de **audioinfo**.

Otros datos como la extensión del archivo lo obtenemos a través de **fileparts** y después vemos si el archivo es .MP3, .WAV u otro archivo que hemos puesto como “Desconocido”.

Luego la potencia la calculamos de forma manual y tras esto vamos aplicando los valores calculados al correspondiente campo de la interfaz que va a ver el usuario.

```
% Obtener información del archivo
info = audioinfo(fullfile(fname));
numChannels = info.NumChannels;
numSamples = info.TotalSamples;
duration = info.Duration;

% Obtener la extensión del archivo cargado
[~, ~, ext] = fileparts(fullfile(fname)); % 'ext' contendrá la extensión del archivo (por ejemplo, '.mp3', '.wav')

% Determinar el tipo de compresión basado en la extensión del archivo
if strcmp(ext, '.mp3')
    compressionMethod = 'MP3'; % Compresión MP3
elseif strcmp(ext, '.wav')
    compressionMethod = 'WAV'; % Compresión WAV
else
    compressionMethod = 'Desconocido'; % En caso de que no sea ni MP3 ni WAV
end

% Calcular potencia
power = sum(abs(audioData(:)))^2 / numSamples;
power_dB = 10 * log10(power);

% Calcular presión sonora
p_ref = 20e-6; % Presión de referencia en pascuales (umbral auditivo)
pressure = sqrt(power_dB) * p_ref;

app.CompressionEditField.EditValue = false;
app.DurationEditField.EditValue = false;
app.FsEditField_2.EditValue = false;
app.PotSonoraEditField.EditValue = false;
app.NumSamplesEditField.EditValue = false;
app.PotencialEditField.EditValue = false;
app.NumChannelsEditField.EditValue = false;
app.FsEditField_2.Value = double(numChannels);
app.FsEditField_2.Value = double(app.Fs);
app.NumSamplesEditField.Value = double(numSamples);
app.CompressionEditField.Value = compressionMethod;
app.DurationEditField.Value = double(duration);
app.PotencialEditField.Value = double(power_dB);
app.PotSonoraEditField.Value = double(pressure);

t = (@length(app.audioData)-1) / app.Fs;
```

Figura 4: Segunda parte función Cargar_Archivo

Compresión	WAV
Nº Canales	1
Fs	2.205e+04
nº muestras	5.361e+05
Duración	24.31
Potencia	0.04036
Pot sonora	4.018e-06

Figura 5: Panel con información del archivo

Esta función se arranca cuando el usuario pulsa el botón de correspondiente (ver Figura 6).

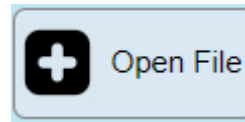


Figura 6: Botón de cargar archivo

3- UpdatePlot

Esta función lo que hace es actualizar la grafica en tiempo real durante la grabación de un audio a través del micrófono pudiendo ver en todo momento la señal que esta captando la aplicación

Como código importante (ver Figura 7) a través de **getaudiodata** vamos a obtener los datos de lo que se esta grabando y luego mediante **drawnow** la gráfica se va actualizando inmediatamente recogiendo y mostrando lo almacenado en Data

Es importante calcular el tiempo para luego mientras se grafica la grabación esta se muestre de forma correcta

```
function updatePlot(app)
    %Esta funcion la vamos a usar para ver lo que se graba a tiempo real

    % Obtener los datos grabados hasta el momento
    Data = getaudiodata(app.recorder);

    % Calcular el tiempo correspondiente
    t = (0:length(Data)-1) / app.recorder.SampleRate;

    % Graficar en tiempo real en los UIAxes
    plot(app.UIAxes3, t, Data);
    axis(app.UIAxes3, 'tight');
    title(app.UIAxes3, 'Grabación en Tiempo Real');
    xlabel(app.UIAxes3, 'Tiempo (s)');
    ylabel(app.UIAxes3, 'Amplitud');
    drawnow; % Actualizar la gráfica inmediatamente
end
```

Figura 7: Función updatePlot

Con esto el usuario puede ver una representación en vivo de la grabación que se está llevando acabo

4- Grabar

Esta función (ver Figura 8) va a iniciar la grabación de audio creando un objeto con **audiorecorder** donde nosotros pasamos los parámetros de frecuencia de muestreo y bits que puede introducir el usuario a través de la interfaz.

Antes de nada, vamos a limpiar las gráficas para el caso en el que haya algún audio cargado y nos de problemas

para después configurar el uso de la función `updatePlot` para ver en tiempo real lo que se está grabando.

Tras todo eso con **record** iniciamos la grabación

```
function Grabar(app, event)
%Callback del boton grabar
% Verificar si ya hay una grabación en curso
if ~isempty(app.recorder) && isrecording(app.recorder)
    uialert(app.UIFigure, 'Ya hay una grabación en curso.', 'Error');
    return;
end

% Obtener parámetros de grabación
fsGrabando = app.fsEditField.Value; % Frecuencia de muestreo desde el campo de texto
bits = str2double(app.BitsDropDown.Value); % Bits desde el DropDown

% Crear objeto audiorecorder
app.recorder = audiorecorder(fsGrabando, bits, 1); % fs: frecuencia, bits: profundidad de bits, 1 canal

% Configurar TimerFcn para actualización en tiempo real
app.recorder.TimerPeriod = 0.1; % Intervalo de actualización en segundos
app.recorder.TimerFcn = @(src, event) updatePlot(app);

% Limpiar gráficas previas
cla(app.UIAxes3);
title(app.UIAxes3, 'Grabación en Proceso...');
xlabel(app.UIAxes3, 'Tiempo (s)');
ylabel(app.UIAxes3, 'Amplitud (V)');
axis(app.UIAxes3, 'tight');

% Iniciar grabación
record(app.recorder);
end
```

Figura 8: Función Grabar

Esta función se arranca cuando el usuario pulsa el botón correspondiente (ver Figura 9)

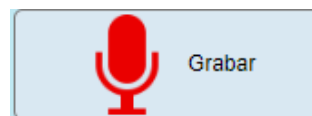


Figura 9: Botón Grabar

5- Parar_Grabación

Con esta función para la grabación que se está en curso y que se muestre ajustada en el dominio del tiempo y su espectrograma.

Para esta función (ver Figura 10), después de comprobar que por lo menos haya alguna grabación en curso paramos la grabación mediante **stop** y obtenemos los datos grabados y los almacenamos para que si luego los modificamos no perdamos los originales.

Tras esto ya solo queda mostrar la señal en el dominio del tiempo y su espectrograma al usuario.

```

function Parar_Grabación(app, event)
%Callback del botón de stop grabación

% Verificar si hay una grabación en curso
if isempty(app.recorder) || ~isrecording(app.recorder)
    uialert(app.UIFigure, 'No hay ninguna grabación en curso.', 'Error');
    return;
end

% Detener la grabación
stop(app.recorder);

% Obtener los datos grabados
app.audioData = getaudiodata(app.recorder);
app.fs = app.recorder.SampleRate;

% Calcular el tiempo
t = (0:length(app.audioData)-1) / app.fs;

% Almacenamos los datos de la señal
app.OriginalSignal = app.audioData; % Señal de audio grabada
app.OriginalTime = t; % Vector de tiempo grabado

% Mostrar en UIAxes del Panel 2
plot(app.UIAxes3, t, app.audioData);
title(app.UIAxes3, 'Audio Grabado');
xlabel(app.UIAxes3, 'Tiempo (s)');
ylabel(app.UIAxes3, 'Amplitud (V)');

% Calcula el espectrograma
signal = app.OriginalSignal; % Convierte estéreo a mono
window = hamming(1024); % Ventana de 512 puntos
noverlap = 512; % Solapamiento entre ventanas
nfft = 2048; % Número de puntos FFT
[~, F, T, P] = spectrogram(signal, window, noverlap, nfft, app.fs);

% Visualiza el espectrograma
imagesc(app.UIAxes2, T, F, 10*log10(P)); % Muestra el espectrograma en dB
colormap(app.UIAxes2);
axis(app.UIAxes2, 'tight');
xlabel(app.UIAxes2, 'Tiempo (s)');
ylabel(app.UIAxes2, 'Frecuencia (Hz)');
app.UIAxes2.YDir = "normal";

```

Figura 10: Función Parar_Grabación

6- Reproducir

En esta función el usuario tras pulsar el botón correspondiente (ver Figura 11) empezara la reproducción del audio que este cargado en la interfaz ya sea una grabación, un archivo de audio cargado o una señal simple generada.



Figura 11: Botón de reproducción

Dentro de esta función (ver Figura 12) comprobamos primero que existe un audio cargado para reproducir en **app.audiodata**, luego tenemos que conocer si tenemos alguna pista de audio pausada y debemos retomar la reproducción donde se pauso mediante **resume(app.player)**.

Tras estas comprobaciones si tenemos algún audio cargado y no ha sido pausado creamos el reproductor mediante **audioplayer** introduciendo la pista de audio y la frecuencia de muestreo almacenadas, y iniciamos la reproducción con **play(app.player)**.


```

function Reproducir(app, event)
    %Callback del botón de reproducir
    % Verificar si hay un archivo de audio cargado
    if isempty(app.audioData)
        uialert(app.UIFigure, 'No hay ningún archivo de audio cargado para reproducir.', 'Error');
        return;
    end

    % Verificar si el objeto player ya existe y está pausado
    if ~isempty(app.player) && isa(app.player, 'audioplayer') && ~isplaying(app.player)
        resume(app.player); % Retomar la reproducción desde donde se pausó
    else
        % Crear el reproductor y reproducir desde el inicio
        app.player = audioplayer(app.audioData, app.fs);
        play(app.player);
    end
end

```

Figura 12: Función Reproducir

7- Pausa

Esta función procede a pausar el audio que esta en reproducción en el momento que el usuario pulsa el botón correspondiente (ver Figura 13)



Figura 13: Botón de pausa

Para esta función (ver Figura 14) hemos comprobado primero que hay un audio para reproducir y luego tras comprobar que el audio se está reproduciendo y no está vacío pausamos la reproducción con **pause(app.player)**

```

function Pausa(app, event)
    %Callback del botón de pausa
    if isempty(app.audioData)
        uialert(app.UIFigure, 'No hay ningún archivo de audio cargado para reproducir.', 'Error');
        return;
    end

    if ~isempty(app.player) && isplaying(app.player)
        pause(app.player); % Pausar la reproducción
    end
end

```

Figura 14: Función Pause

8- Stop

Esta función procede a detener el audio que esta en reproducción y reiniciar el reproductor desde el inicio cuando el usuario pulsa el botón correspondiente (ver Figura 15)

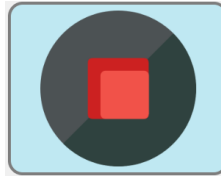


Figura 15: Botón de stop

Dentro de esta función (ver Figura 16) tras comprobar que hay algún audio reproduciendo detenemos la reproducción con **stop(app.player)** y por ultimo reiniciamos el reproductor para que empiece desde cero mediante **audioplayer**.

```
function Stop(app, event)
%Callback del botón de parar
if ~isempty(app.player) && isplaying(app.player)
    stop(app.player); % Detiene la reproducción
end

% Reiniciar el reproductor creando un nuevo objeto audioplayer
app.player = audioplayer(app.audioData, app.fs);
end
```

Función 16: Función Stop

9- Señal_Generada

Esta función va a generar señales simples mediante los parámetros que introduce el usuario (ver Figura 17), dentro de los tipos de señales tenemos el seno, coseno, cuadrática, triangular y dientes de sierra

Función 17: Parámetros

Dentro de esta función (ver Figura 18.1) primero vamos a recoger los parámetros que introduce el usuario a través de la interfaz. Tras esto calculamos los tiempos a través de las frecuencias introducidas.

Con un **switch** vamos a comprobar que tipo de señal ha seleccionado el usuario y dependiendo de la elección vamos a definir nuestra señal (signal), luego después de tener todos los parámetros antes de graficarlos vamos a asignarlos a las propiedades **app.audioData** y **app.fs** y los vamos a almacenar en **app.OriginalSignal** y **app.OriginalTime** para no perder las propiedades en caso de alguna posterior modificación.

```
function Señal_Generada(app, event)
    signalType = app.SealDropDown.Value;
    amplitude = app.AmplitudEditField.Value;
    app.frecuencia = app.FcEditField.Value;
    fsample = app.FsEditField_3.Value;
    periodos = app.PeriodosEditField.Value;

    %Calcula la duracion total en funcion de los periodos y f
    T = 1/app.frecuencia;
    tTotal = periodos*T;
    tVector = 0:1/fsample:tTotal;

    %Generar la señal seleccionas
    switch signalType
        case 'Seno'
            signal = amplitude * sin(2*pi*app.frecuencia * tVector);
        case 'Coseno'
            signal = amplitude * cos(2*pi*app.frecuencia * tVector);
        case 'Cuadrada'
            signal = amplitude * square(2*pi*app.frecuencia * tVector);
        case 'Triangular'
            signal = amplitude * sawtooth(2*pi*app.frecuencia * tVector, 0.5);
        case 'Dientes'
            signal = amplitude * sawtooth(2*pi*app.frecuencia * tVector);
        otherwise
            uialert(app.UIFigure, 'Tipo de señal no reconocida')
            return;
    end

    % Asignar señal generada a 'app.audioData' y otras propiedades
    app.audioData = signal(:); % Asegurar que sea un vector columna
    app.fs = fsample;
    app.OriginalSignal = app.audioData; % Almacenar como señal original
    app.OriginalTime = tVector(:); % Almacenar el tiempo original
    signallength = length(signal);
end
```

Figura 18.1: Función Señal_Generada

Por último, vamos a graficarlo (ver Figura 18.2) tras mostrar la señal en el dominio del tiempo vamos a calcular el espectrograma, para esto hemos tenido que comprobar la longitud de la ventana adecuada a la duración de la señal que se ha escogido debido a que si la duración es muy pequeña da error.

Luego vamos a crear el espectrograma a través de la función **spectrogram** y lo graficamos.

```

%Graficamos
plot(app.UIAxes3, tVector, signal);
axis(app.UIAxes3, 'tight');
title(app.UIAxes3, ['Señal: ', signalType])

windowSize = 1024;

if signalLength < windowSize
    windowSize = signalLength; % Ajustar a la longitud de la señal
end
window = hamming(windowSize); % Crear ventana ajustada
noverlap = max(1, round(windowSize / 2)); % Solapamiento (mínimo 1)
nfft = max(256, windowSize); % Número de puntos FFT ajustado
% Comprobar si la señal es suficientemente larga para calcular el espectrograma
if signalLength < 2
    uialert(app.UIFigure, 'La señal generada es demasiado corta para calcular el espectrograma.', 'Error');
    return;
end

[~, F, T, P] = spectrogram(signal, window, noverlap, nfft, app.frecuencia, 'yaxis');

% Visualiza el espectrograma
imagesc(app.UIAxes2, T, F, 10*log10(P)); % Muestra el espectrograma en dB
colormap(app.UIAxes2);
axis(app.UIAxes2, 'tight');
xlabel(app.UIAxes2, 'Tiempo (s)');
ylabel(app.UIAxes2, 'Frecuencia (Hz)');
app.UIAxes2.YDir = "normal";

end

```

Figura 18.2: Función Señal_Generada

10- Invertir_Senal

Esta función procede a la inversión del eje temporal a través de la activación de un ‘Check Box’ por parte del usuario (ver Figura 19)

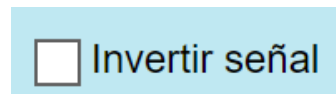


Figura 19: Check Box

Primero (ver Figura 20) vamos a recoger el valor del “Check Box” para ver si esta activado o no. Si esta activado vamos a detener la reproducción y luego invertimos el eje de tiempos y la señal con **flipud**, tras esto reiniciamos el reproductor.

Además del mismo modo hemos calculado el espectrograma correspondiente para que también se invierta en caso de activar la función

Si se desactiva vamos a restaurar la señal mediante los parámetros **app.OriginalSignal** y **app.OriginalTime** y reiniciamos el reproductor.

```

function Invertir_Senal(app, event)
    activado = app.InvertirsealCheckBox.Value;

    if activado
        stop(app.player); % Detiene la reproducción
        % Invertir la señal en el eje de tiempos
        invertedTime = -app.OriginalTime; % Inversión del eje de tiempos
        app.audioData = flipud(app.OriginalSignal); % Señal invertida en el dominio temporal
        plot(app.UIAxes3, invertedTime, app.OriginalSignal);

        % Reiniciar el reproductor creando un nuevo objeto audioplayer
        app.player = audioplayer(app.audioData, app.fs);
    else
        stop(app.player); % Detiene la reproducción
        % Restaurar la señal original
        app.audioData = app.OriginalSignal; % Restaurar la señal original
        plot(app.UIAxes3, app.OriginalTime, app.OriginalSignal);
        axis(app.UIAxes3, 'tight');

        % Reiniciar el reproductor creando un nuevo objeto audioplayer
        app.player = audioplayer(app.audioData, app.fs);
    end
end

```

Figura 20: Función Invertir_Senal

11- Histograma

Con esta función se mostrará una nueva ventana, cuando el usuario accione el botón (ver Figura 21), con el histograma correspondiente a la última señal cargada pueda ser editada o no.

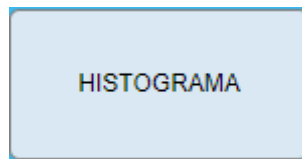


Figura 21: Botón histograma

Para esta función (ver Figura 22) tras comprobar que tenemos un audio cargado creamos la nueva ventana con **uifigure** y el nuevo axes con **uiaxes**.

Para el calculo del histograma realizamos la FFT a través de **fft** y **fftshift** y por último lo graficamos en el nuevo axes que hemos creado.

```

function Histograma(app, event)
    % Verificar si hay un audio cargado
    if isempty(app.audioData)
        uialert(app.UIFigure, 'No hay ningún archivo de audio cargado.', 'Error');
        return;
    end

    % Crear una nueva ventana para el histograma
    figHistograma = uifigure('Name', 'Histograma del Audio', 'Position', [100, 100, 600, 400]);

    % Crear el UIAxes en la nueva ventana
    axHistograma = uiaxes('Parent', figHistograma, 'Position', [50, 50, 500, 300]);

    % Parámetros de la FFT
    Fs = app.fs; % Frecuencia de muestreo (debe estar disponible en app)
    nfft = 1024; % Tamaño de la FFT
    L = length(app.audioData); % Longitud de la señal

    % Calcular la FFT de la señal
    fft_signal = fft(app.audioData, nfft); % FFT de la señal

    % Aplicar fftshift para centrar las frecuencias en torno a 0
    fft_shifted = fftshift(fft_signal); % Mueve las frecuencias negativas al centro
    frequencies = linspace(-Fs/2, Fs/2, nfft); % Eje de frecuencias con fftshift

    % Calcular la potencia
    P = abs(fft_shifted / L).^2; % Potencia normalizada

    % Graficar el espectro en dB
    plot(axHistograma, frequencies, P, 'LineWidth', 1.5);
    title(axHistograma, 'FFT');
    xlabel(axHistograma, 'Frecuencia (Hz)');
    ylabel(axHistograma, 'Potencia');
    grid(axHistograma, 'on');
end

```

Figura 22: Función Histograma

12- CompresionExpansion

Con esta función tras la introducción de un facto k introducido por el usuario (ver Figura 23) se procede a hacer una expansión o compresión de la señal.

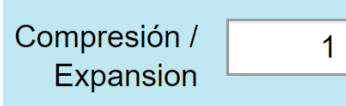


Figura 23: Parámetro k

Dentro de la función (ver Figura 24) tras comprobar que tenemos una función cargada recogemos el valor introducido por el usuario. Luego vamos a multiplicar el tiempo original por el factor introducido y graficamos el nuevo dominio temporal y el nuevo espectrograma

```
function CompresionExpansion(app, event)
% Verificar si hay una señal cargada
if isempty(app.audioData)
    uialert(app.UIFigure, 'No hay señal cargada para modificar.', 'Error');
    return;
end

% El valor de k factor K
K = app.CompresionExpansionEditField.Value;

% Aplicar compresión o expansión temporal
tOriginal = app.OriginalTime; % Tiempo original de la señal
tNuevo = tOriginal * K; % Escalado del tiempo (compresión/expansión)

% Interpolación de la señal para ajustar al nuevo tiempo
app.audioData = interp1(tOriginal, app.OriginalSignal, tNuevo, 'linear', 0);

% Graficar la señal modificada
plot(app.UIAxes3_2, tNuevo, app.audioData);
title(app.UIAxes3_2, sprintf('Señal Modificada (K = %.2f)', K));
xlabel(app.UIAxes3_2, 'Tiempo (s)');
ylabel(app.UIAxes3_2, 'Amplitud');
axis(app.UIAxes3_2, 'tight');

% Actualizar el reproductor de audio
app.player = audioplayer(app.audioData, app.fs);

signal = app.audioData;
window = hamming(1024); % Ventana de 512 puntos
noverlap = 512; % Solapamiento entre ventanas
nfft = 2048; % Número de puntos FFT
[~, F, T, P] = spectrogram(signal, window, noverlap, nfft, app.fs);

% Visualiza el espectrograma
imagesc(app.UIAxes2_2, T, F, 10*log10(P)); % Muestra el espectrograma en dB
colormap(app.UIAxes2_2);
axis(app.UIAxes2_2, 'tight');
xlabel(app.UIAxes2_2, 'Tiempo (s)');
ylabel(app.UIAxes2_2, 'Frecuencia (Hz)');
app.UIAxes2_2.YDir = "normal";
end
```

Figura 24: Función *CompresionExpansion*

13- Efectos

Con esta función el usuario puede elegir entre varios efectos disponibles (ver Figura 25) para aplicar a la señal cargada.

Figura 25: Efectos

Dentro de esta función (ver Figura 26.1) tras comprobar que tenemos alguna señal cargada y recoger los parámetros originales vamos a ver que elección a introducido el usuario mediante un **switch** y dependiendo de la selección vamos a modificar el valor de **app.audioData**.

```
function Efectos(app, event)
% Verificar si hay audio cargado o generado
if isempty(app.audioData)
    uialert(app.UIFigure, 'No hay señal cargada o generada.', 'Error');
    return;
end

% Restaurar la señal original antes de aplicar un nuevo efecto
app.audioData = app.OriginalSignal;

% Determinar el efecto seleccionado
selectedEfecto = app.EfectosButtonGroup.SelectedObject.Text;

% Aplicar el efecto seleccionado
switch selectedEfecto
case 'Eco'
    % Crear un eco básico
    delay = round(0.3 * app.fs); % Retardo de 300 ms
    atten = 0.6; % Atenuación del eco
    ecoSignal = [app.audioData; zeros(delay, 1)]; % Extender señal con ceros
    ecoSignal(delay+1:end) = ecoSignal(delay+1:end) + atten * app.audioData;
    app.audioData = ecoSignal;
case 'Distorsion'
    % Distorsión mediante saturación
    gain = 5; % Ganancia
    app.audioData = tanh(gain * app.audioData); % Saturación con tangente hiperbólica
case 'Reverberacion'
    % Reverberación simulada con un filtro de convolución
    reverbkernel = [1; zeros(2000, 1); 0.5; zeros(2000, 1); 0.25]; % Kernel de reverberación
    app.audioData = conv(app.audioData, reverbkernel, 'same');
case 'Amplificacion'
    % Amplificación básica
    app.audioData = app.audioData * 3; % Duplicar amplitud
case 'Ninguno'
    % Sin efecto (se mantiene la señal original)
    app.audioData = app.OriginalSignal;
otherwise
    % Si por alguna razón no se reconoce el efecto
    app.audioData = app.OriginalSignal;
end
```

Figura 26.1 Funcion Efectos

Para el efecto “Eco” hemos introducido un delay y luego se sumará a la señal original atenuada mediante una atenuación que hemos introducido con anterioridad.

Para el efecto “Distorsión” hemos aplicado una saturación mediante la función **tanh** que comprime las amplitudes grandes haciendo una simulación de “saturación”

Para el efecto “Reverberación” se simula un eco prolongado con un filtro de convolución utilizando una respuesta al impulso con retardos y atenuaciones

Para el efecto “Amplificación” se multiplica la señal por un factor constante aumentando la amplitud

Tras todo esto (ver Figura 26.2) procedemos a la creación del nuevo objeto con la modificación necesaria y lo graficamos, como también se le puede introducir efectos a la señal generada debemos comprobar el tamaño de la ventana para ver el espectrograma por si es muy corta evitar errores

```
% Crear un nuevo objeto audioplayer con la señal modificada
app.player = audioplayer(app.audioData, app.fs);

% Graficar la señal con el efecto aplicado en UIAxes3_2
t = (0:length(app.audioData)-1) / app.fs;
plot(app.UIAxes3_2, t, app.audioData);
axis(app.UIAxes3_2, 'tight');
title(app.UIAxes3_2, ['Señal con ', selectedEffect]);
xlabel(app.UIAxes3_2, 'Tiempo (s)');
ylabel(app.UIAxes3_2, 'Amplitud');

% Actualizar el espectrograma en UIAxes2_2
% Actualizar el espectrograma en UIAxes2_2
if length(app.audioData) < 512
    uiauter(app.UIFigure, 'La señal es demasiado corta para calcular el espectrograma. Intente con una señal más larga.', 'advertencia');
    return;
end
window = hamming(1024);
noverlap = 512;
nfft = 2048;
[P, F, T, P] = spectrogram(app.audioData, window, noverlap, nfft, app.fs);
imagesc(app.UIAxes2_2, T, P, 10*log10(P));
colorbar(app.UIAxes2_2, 'jet');
axis(app.UIAxes2_2, 'tight');
xlabel(app.UIAxes2_2, 'Tiempo (s)');
ylabel(app.UIAxes2_2, 'Frecuencia (Hz)');
app.UIAxes2_2.VDir = 'normal';
end
```

Figura 26.2: Función Efectos

14- Ruido

Con esta función el usuario puede elegir entre varios ruidos disponibles (ver Figura 27) para aplicar a la señal cargada.



Ruido

- ☒ Ninguno
- ☐ Ruido Blanco
- ☐ Ruido Uniforme
- ☐ Ruido Impulsivo
- ☐ Ruido Sinusoidal

Figura 27: Ruidos

Para esta función (ver Figura 28.1) tras comprobar que tenemos alguna señal cargada y recoger los parámetros originales vamos a ver que elección a introducido el

usuario mediante un **switch** y dependiendo de la selección creamos un ruido

```
function Ruido(app, event)
% Verificar si hay audio cargado o generado
if isempty(app.audioData)
    uialert(app.UIFigure, 'No hay señal cargada o generada.', 'Error');
    return;
end

% Determinar el efecto seleccionado
selectedRuido = app.RuidoButtonGroup.SelectedObject.Text;

% Restaurar la señal original antes de agregar ruido
app.audioData = app.OriginalSignal;

% Aplicar el efecto seleccionado
switch selectedRuido
case 'Ruido Blanco'
    % Ruido blanco
    ruido = 0.05 * randn(size(app.audioData));
case 'Ruido Uniforme'
    % Ruido uniforme (rango -0.025 a 0.025)
    ruido = 0.05 * (rand(size(app.audioData)) - 0.5);
case 'Ruido Impulsivo'
    % Ruido impulsivo con picos aleatorios
    ruido = zeros(size(app.audioData));
    numSpikes = round(length(app.audioData) * 0.01); % 1% de la señal
    spikeIndices = randperm(length(app.audioData), numSpikes);
    ruido(spikeIndices) = 0.5 * sign(randn(size(spikeIndices))); % Picos +/- 0.5
case 'Ruido Sinusoidal'
    % Ruido sinusoidal con frecuencia aleatoria
    t = (0:length(app.audioData)-1) / app.fs;
    freq = 50 + 200 * rand; % Frecuencia entre 50 Hz y 250 Hz
    ruido = 0.05 * sin(2 * pi * freq * t(:));
case 'Ninguno'
    % Sin ruido
    ruido = zeros(size(app.audioData));
otherwise
    % Sin ruido
    ruido = zeros(size(app.audioData));
end
```

Figura 28.1: Función Ruido

Para el ruido “Blanco” hemos generado el ruido con una distribución normal ya que el ruido blanco presenta una potencia uniforme frecuencial.

Para el ruido “Uniforme” hemos generado valores aleatorios uniformemente distribuidos similar a lo realizado en el ruido blanco

Para el ruido “Impulsivo” introducimos picos esporádicos en la señal generando valores aleatorios simulando “frecuencias bruscas”

Para el ruido “Sinusoidal” genera una señal sinusoidal que se sumara a la original introduciendo un tono sinusoidal con frecuencia constante

Por último (ver Figura 26.2) procedemos a la creación del nuevo objeto sumándole al original el ruido y lo graficamos, como también se le puede introducir ruidos a la señal generada debemos comprobar el tamaño de la ventana para ver el espectrograma por si es muy corta evitar errores.

```

% Agregar el ruido a la señal existente
app.audioData = app.audioData + ruido;

% Crear un nuevo objeto audioplayer con la señal modificada
app.player = audioplayer(app.audioData, app.fs);

% Graficar la señal con ruido en UIAxes3_2
t = (0:length(app.audioData)-1) / app.fs;
plot(app.UIAxes3_2, t, app.audioData);
axis(app.UIAxes3_2, 'tight');
title(app.UIAxes3_2, 'Señal con ', selectedRuido);
xlabel(app.UIAxes3_2, 'Tiempo (s)');
ylabel(app.UIAxes3_2, 'Amplitud');

% Actualizar el espectrograma en UIAxes2_2
if length(app.audioData) < 512
    uiaert(app.UIFigure, 'La señal es demasiado corta para calcular el espectrograma. Intente con una señal más larga.', 'Advertencia');
    return;
end
window = hamming(512);
overlap = 512;
efft = 2048;
[~, F, T, P] = spectrogram(app.audioData, window, overlap, efft, app.fs);
imagesc(app.UIAxes2_2, T, P, 10*log10(P));
colormap(app.UIAxes2_2, 'jet');
axis(app.UIAxes2_2, 'tight');
xlabel(app.UIAxes2_2, 'Tiempo (s)');
ylabel(app.UIAxes2_2, 'Frecuencia (Hz)');
app.UIAxes2_2.VDir = 'normal';

end

```

Figura 28.2: Función Ruido

15- CambiarPanel

Con esta función el usuario cuando pulse el botón (ver Figura 29) puede visualizar la señal cargada en el panel inferior.

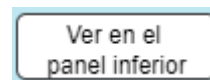


Figura 29: Botón CambiarPanel

Para esta función lo que hemos hecho es obtener los datos de los dos gráficos superiores para replicarlos abajo mediante **findobj** debido a que así si se modificase la señal la cual se carga en los gráficos inferiores nos aseguramos de que al pulsar el botón lo que se visualice sea la señal que se ve en los gráficos superiores.

Una vez obtenido los datos los hemos graficado en los “axes” correspondientes.

PEGAR CODIGO