|  |
| --- |
| Memoria de Programación |

Índice

[Introducción 2](#_Toc26725712)

[Propiedades 3](#_Toc26725713)

[Gráficas Tiempo/Frecuencia 4](#_Toc26725714)

[Gráfica Tiempo 4](#_Toc26725715)

[Gráficas Frecuencia 4](#_Toc26725716)

[-Transformada de Fourier 5](#_Toc26725717)

[-Espectrograma 6](#_Toc26725718)

[Genera Señales 6](#_Toc26725719)

[Grabación 6](#_Toc26725720)

[Señales Simples 8](#_Toc26725721)

[-Seno 10](#_Toc26725722)

[-Coseno 10](#_Toc26725723)

[-Cuadrada 10](#_Toc26725724)

[-Triangular 10](#_Toc26725725)

[-Dientes de sierra 11](#_Toc26725726)

[-Chirp lineal desde 0 11](#_Toc26725727)

[-Chirp cuadrático 11](#_Toc26725728)

[-Constante 12](#_Toc26725729)

[-Lineal 12](#_Toc26725730)

[Piano 13](#_Toc26725731)

[Ruido 15](#_Toc26725732)

[Cargar/Guardar Archivo 16](#_Toc26725733)

[Cargar Archivo 16](#_Toc26725734)

[Guardar Archivo 17](#_Toc26725735)

[Modificar Audio 18](#_Toc26725736)

[Velocidad/Inversión 19](#_Toc26725737)

[Efectos 21](#_Toc26725738)

[-Eco 21](#_Toc26725739)

[-Flanger 22](#_Toc26725740)

[-Wah-wah 22](#_Toc26725741)

[-Fuzz 23](#_Toc26725742)

[-Trémolo 23](#_Toc26725743)

[Cuantificación 24](#_Toc26725744)

[Aplicar ruido 24](#_Toc26725745)

[Filtros 24](#_Toc26725746)

[Modulación/Demodulación 26](#_Toc26725747)

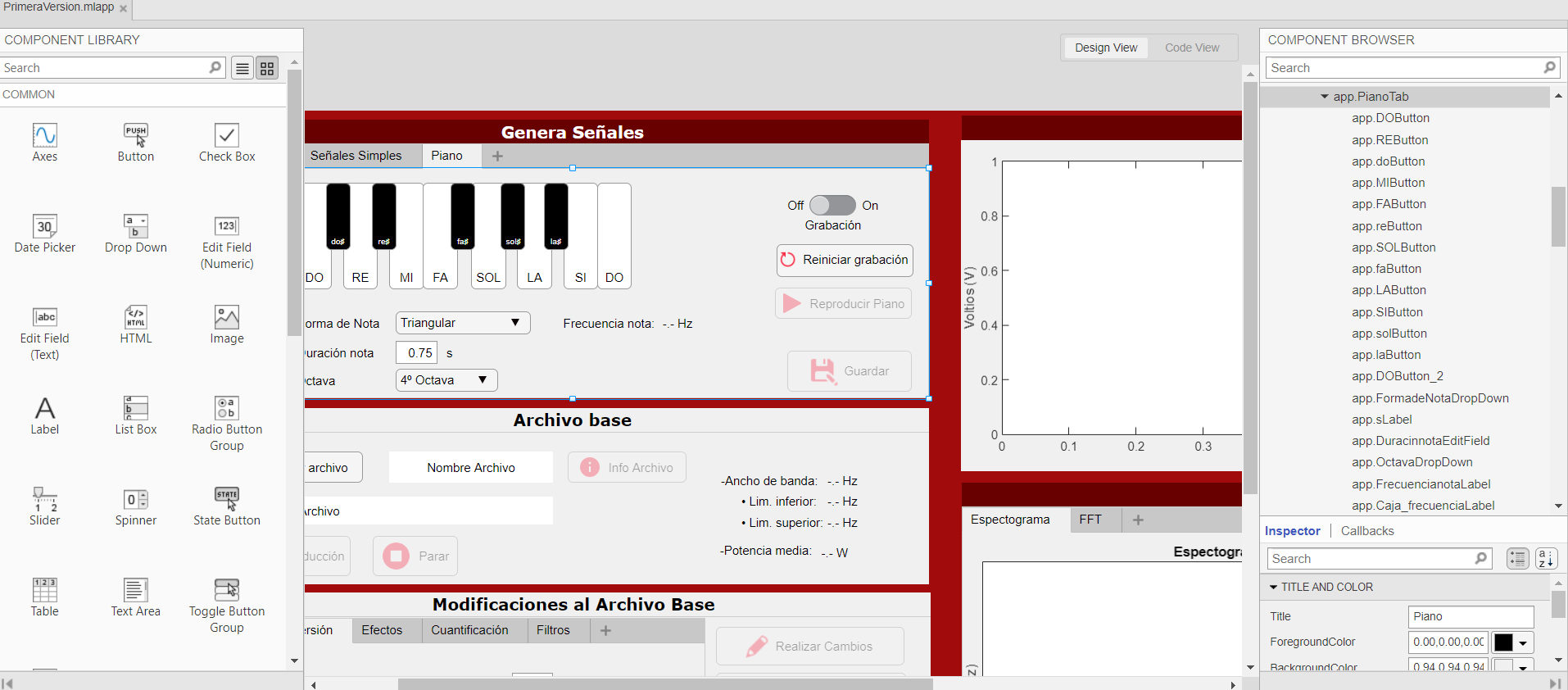
[Referencias 26](#_Toc26725748)

# Introducción

Para la realización de esta aplicación en Matlab se ha hecho uso de la interfaz que proporciona este programa, *App Designer*, un entorno para crear apps con interfaces gráficas de usuario (GUI). La versión con la que se ha creado esta app es la de MATLAB R2019b.

Esta GUI permite diferenciar de forma paralela el código usado para el estilo gráfico de cada elemento y el código que realiza las funciones usando parámetros introducidos por esos elementos y mostrando el resultado de igual forma gracias a estos.

**En cuanto, a la “design view”:**



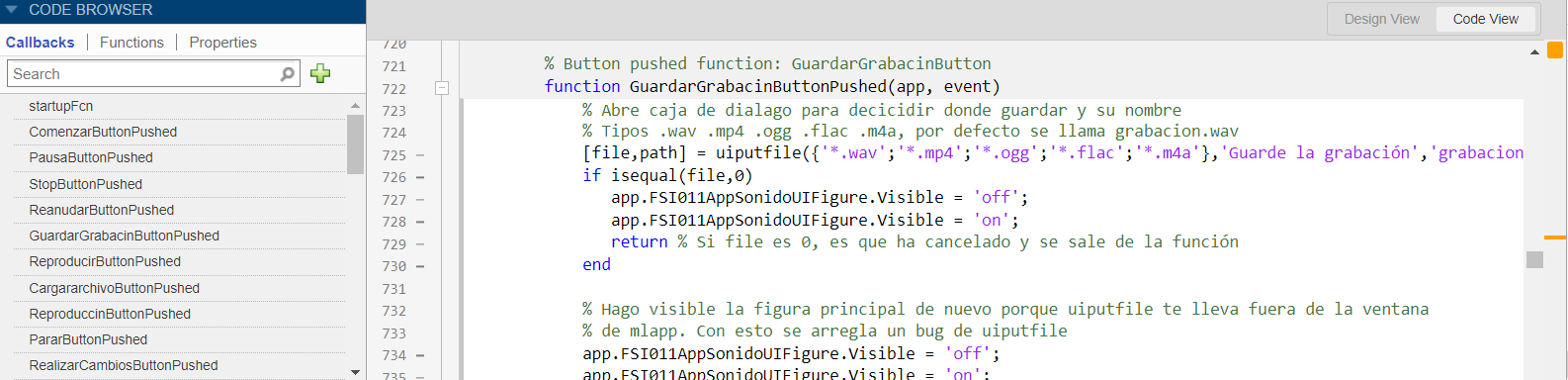
Lo principal de esta GUI es el uso de 4 paneles diferenciados:

1. Genera Señales
2. Archivo Base
3. Modificaciones al Archivo Base
4. Gráficas

Cada uno de estos, hace uso de “paneles” que dividen claramente la interfaz general, con los paneles de ‘Genera Señales’ y ‘Modificaciones al Archivo Base’ además se incluye un “tab Group” que permite, a través de pestañas, entrar en subventanas donde introducir/modificar distintos parámetros ya sea a la hora de generar nuevas señales o introducir distintos efectos.

En cuanto a las gráficas se ha divido en 2 paneles una mostrando señales en el tiempo, y otro panel para las de frecuencia, que además divide gracias a un “tab Group”, el espectrograma y la transformada de Fourier. Y para poder ver la señal original y modificada se han creado otros dos paneles iguales y superpuestos “invisibles” (hasta que se pide mostrarlos) que tienen 2 gráficas cada uno.

**En cuanto, a la “code view”:**



Lo más relevante de esta parte son el uso de callbacks y funciones, los callbacks son funciones que se llaman al realizar el usuario algún tipo de interacción con los botones y demás de la GUI, mientras que las funciones son instrucciones que pueden llamar las funciones callback. Y para el uso de variables compartidas entre todas las funciones y callback, existen las propiedades a las que se pueden acceder desde todo el programa.

Para la explicación de la programación de las distintas funcionalidades de nuestra app, hablaremos primero de las propiedades de las que consta, de las funciones de los distintos apartados de Gráficas Tiempo/Frecuencia, Generación Señales, Cargar Archivo y Modificar Audio.

# Propiedades

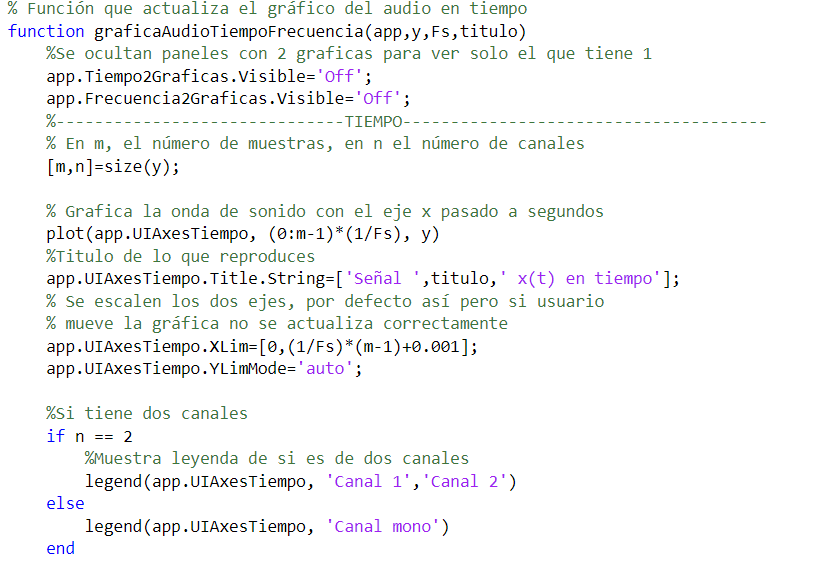
Se explicarán las propiedades usadas por la app:

* grabacionObj: Objeto de tipo audiorecord que guarda la grabación
* audioSeleccionado: Objeto de tipo audioplayer que guarda el audio seleccionado, necesario para reproducirlo
* audioSamplesSeleccionado: Guarda los valores de sample de audio seleccionado, necesarios para modificarlos
* audioModificado: Objeto de tipo audioplayer que guarda el audio modificado, necesario para reproducirlo
* audioSamplesModificado: Guarda los valores de sample de audio modificado
* oldpath = addpath('iconos'): Necesario para poder guardar los iconos usados en una carpeta debido a un bug de AppDesigner que no deja seleccionar rutas relativas y se debe añadir al workspace la carpeta iconos [1].
* infoAudioSel: Objeto info del audio seleccionado
* ySenalSencilla: Muestras de la señal sencilla generada
* fsSenalSencilla: Fs de las muestras de la señal sencilla creada
* yGrabacionPiano: Muestras de la señal guardada de piano a Fs 44100
* ySenalRuido: Muestras de la señal de ruido generada
* fsSenalRuido: Fs de las muestras de la señal de ruido creada

# Gráficas Tiempo/Frecuencia

Para mostrar las señales en tiempo y frecuencia se usan las funciones graficaAudioTiempo Frecuencia (app, y, Fs, titulo) y graficaAudioTiempoFrecuenciaOrigYModif(app, yOr, FsOr, yMod, FsMod). A la primera hay que pasar la app, la señal de audio, su frecuencia de muestro Fs y el título de lo que vas a graficar (Piano, grabación, etc..). Y como respuesta muestra la señal enviada en los paneles en tiempo y frecuencia. La segunda es igual, pero le pasas 2 señales y muestra las señales en 2 gráficas separadas, una la original y otra la modificada. Dado que esta última función hace uso de la primera y se basa en lo mismo, solo se explicará el código de graficaAudioTiempoFrecuencia(app, y, Fs, titulo).

## Gráfica Tiempo



Para mostrar en tiempo primero con size(), se obtiene el número de canales y de muestras, con el número de muestras m se pasan a tiempo con (0:m-1)\*(1/Fs), ya que desde la muestra 1 en t=0, el resto de muestras se encuentran en Ts=(1/Fs). Si hay dos canales se muestran los dos canales en la misma gráfica. Después se cambia el título del plot según el string título que se pase, luego con XLim y YLim se ajustan los ejes. Por último, con el número de canales n, se muestra la leyenda indicando si es estéreo o si es mono.

## Gráficas Frecuencia

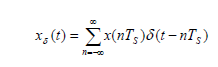
Solo veremos las gráficas de frecuencia de canales mono, pasamos de 2 canales a 1, si hiciese falta con la media de los canales, haciendo uso de sum() para sumarlos y del tamaño para dividir.

% canales a 1, si es 1 no se modifica

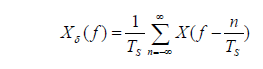
y = sum(y, 2) / size(y, 2);

### -Transformada de Fourier

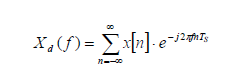
Una vez que definidas las señales en el tiempo, se va a calcular el espectro de las mismas, para lo cual se hará uso de la transformada de Fourier. Se ha de tener en cuenta que en Matlab® se trabaja con muestras discretas: x[n], es decir, con secuencias discretas de N muestras, separadas TS (período de muestreo). El primer paso para obtener una señal de este tipo consiste en tomar muestras de la señal continua en instantes de tiempo separados TS [2]:



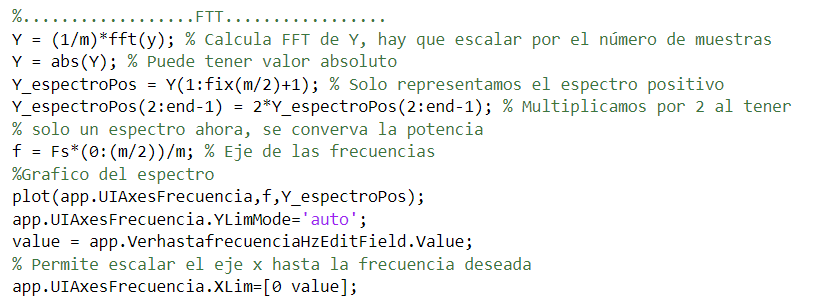
Si se calcula la transformada de Fourier de la misma se obtiene:



En el caso de que se parta directamente de una secuencia discreta, x[n], como ocurre en Matlab®, la expresión de la transformada de Fourier discreta (DFT) es:

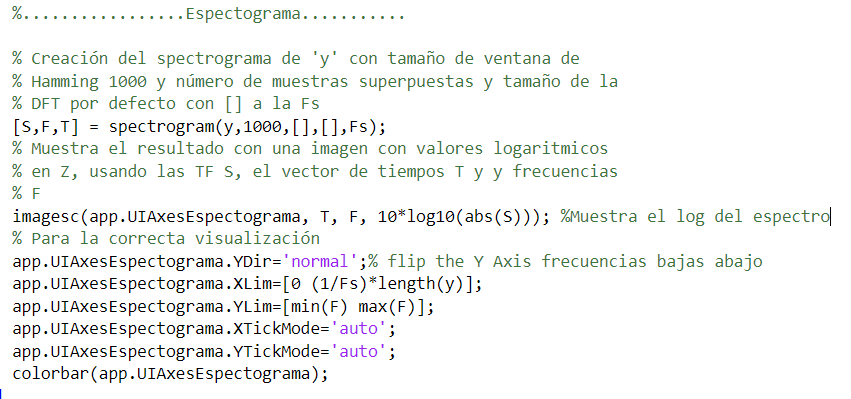


Para reducir la carga computacional asociada, Matlab® implementa la DFT mediante el algoritmo de la transformada rápida de Fourier (FFT). El comando correspondiente es fft, que suele combinarse con la función fftshift. Es de notar, que la frecuencia más alta con significado físico real visible mediante el comando fft es la mitad de la frecuencia de muestreo. Esto hay que tenerlo en cuenta cuando se vayan a representar los espectros.



Como son señales de energía, hay que escalar la FFT por el número de muestras por eso el factor de escala (1/m), además se hace el valor absoluto, ya que tiene componentes complejas y se visualiza el módulo. Después se altera el vector para solo mostrar las frecuencias positivas, puesto que al ser una señal real su espectro es simétrico, al hacer esto se multiplica el espectro por 2, para que la potencia sea la misma. El resto de código es para escalar la gráfica y ver hasta la frecuencia introducida por el usuario.

### -Espectrograma



Para el espectrograma se hace uso de [S,F,T] = spectrogram(y,1000,[],[],Fs); donde y es la señal de audio con 1000, se indica el tamaño de la ventana que divide la señal en segmentos de esa longitud con una ventana de Hamming, con los [], indicas que utilice un número de muestras superpuestas que produce una superposición del 50% entre los segmentos y Fs la frecuencia de muestreo de la señal. S te da la Transformada de Fourier a corto plazo, F y T, los vectores de frecuencia y tiempo. Para dibujarlo usamos imagesc(), al que indicamos las filas y columnas que son T y F, y el valor de cada “pixel” es el valor absoluto de S en escala logarítmica. Además, añadimos un colorbar para saber en qué valores se encuentra a través del color. El resto de las funciones es para el correcto visionado del espectrograma.

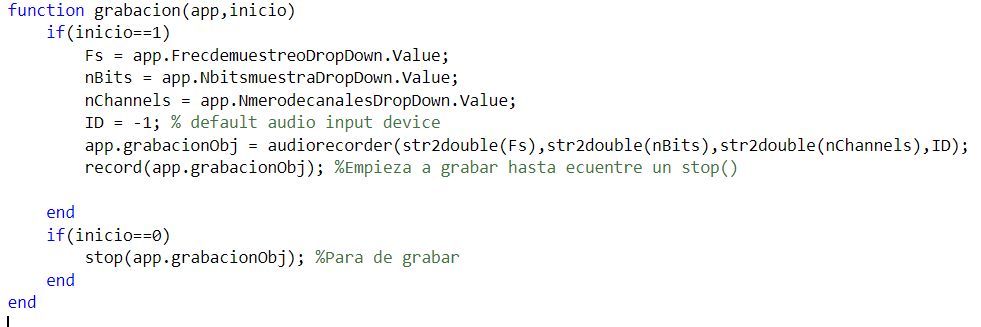
# Genera Señales

Aquí se encuentran las funciones relacionadas con generar distintos tipos de audio.

## Grabación

Para la grabación se hace uso del objeto de tipo audiorecorder, que permite pausarlo, reanudar, parar, etc…

Para empezar y parar de grabar se usa la función grabacion(app,inicio), donde inicio puede valor 1 para empezar a grabar y 0 para parar.



Usa el Fs, nBits y canales que indique el usuario y graba con el dispositivo por defecto. Se crea con audiorecorder(Fs,nBits,NumChannels,ID) y se empieza a grabar con record() y se finaliza con stop(). Para pararlo pause() y reanudar resume(). Implementación a través de callbacks:

% Button pushed function: ComenzarButton

function ComenzarButtonPushed(app, event)

grabacion(app,1)

% Al comenzar se activa el de parar y desactiva el de comenzar

app.StopButton.Enable = 'on';

app.ComenzarButton.Enable = 'off';

% Para ver el bóton de pausa una vez empieces a grabar

app.PausaButton.Visible='on';

end

% Button pushed function: PausaButton

function PausaButtonPushed(app, event)

pause(app.grabacionObj); % Se pausa la grabación

app.PausaButton.Visible='off'; % Se deja de ver este botón de pausa

app.ReanudarButton.Visible='on'; % Se ve el botón para reanudar la grabación

end

% Button pushed function: StopButton

function StopButtonPushed(app, event)

grabacion(app,0)

Fs=app.grabacionObj.SampleRate;

y=getaudiodata(app.grabacionObj);

graficaAudioTiempoFrecuencia(app,y,Fs,' de Grabación');

% Botón de parada se desactica y el de comenzar se activa

app.StopButton.Enable = 'off';

app.ComenzarButton.Enable = 'on';

% Botón de pausa y reanudación se ocultan

app.PausaButton.Visible='off';

app.ReanudarButton.Visible='off';

% Botones de reproduccion y guardado se activan

app.GuardarGrabacinButton.Enable='on';

app.ReproducirButton.Enable='on';

end

% Button pushed function: ReanudarButton

function ReanudarButtonPushed(app, event)

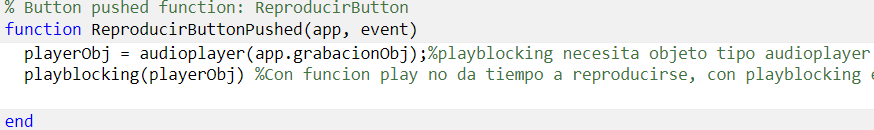
resume(app.grabacionObj); % Se reanuda la grabación

app.ReanudarButton.Visible='off'; % Se ve el botón para parar la grabación

app.PausaButton.Visible='on'; % Se vuelve a ver botón de pausa

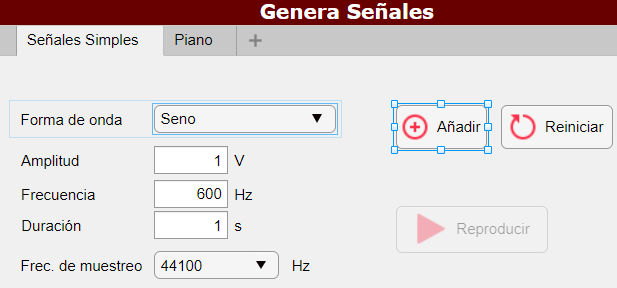
end

Para reproducirlo se usa un callback, este objeto se pasa a uno de tipo audioplayer y se reproduce:

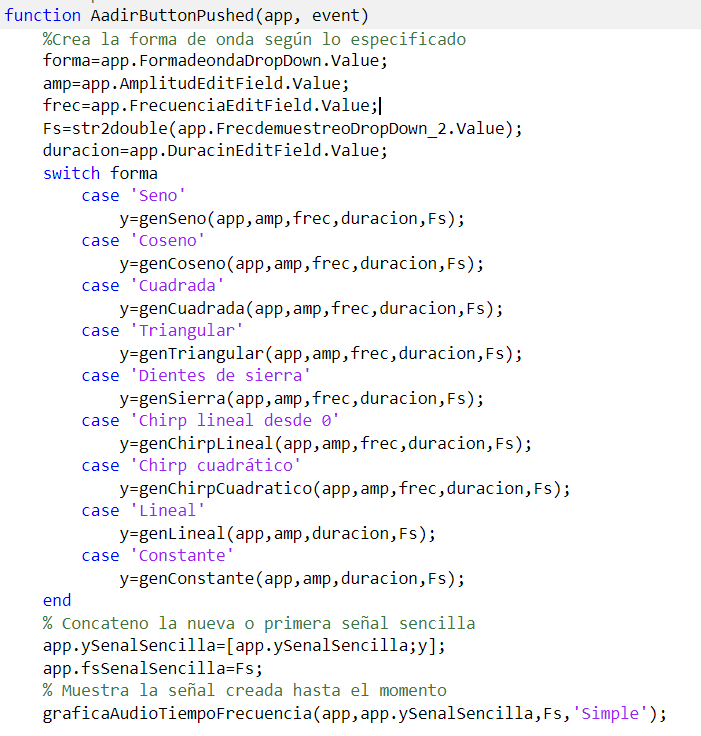


## Señales Simples

Para la generación de señales se parte sobre todo de los callback de los botones de añadir y reiniciar. Añadir usa una serie de funciones que crean las señales especificadas por las opciones que hay para introducir por el usuario. Mientras que reiniciar borra la señal creada hasta el momento.

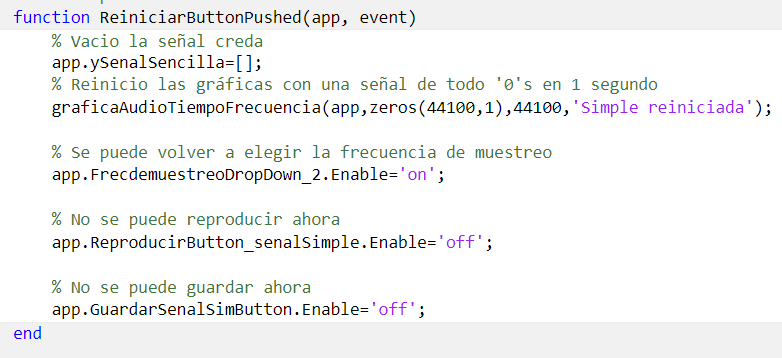


El callback de añadir:



Este callback obtiene los valores introducidos por el usuario y llama a través de un switch, a la función generadora de esas formas de onda con los parámetros indicados. Después se concatena la columna generada con las señales generadas previamente y se muestra su gráfico con la función graficaAudioTiempoFrecuencia().

El botón de reiniciar:



Simplemente vacía la señal creada y muestra en las gráficas un segundo de ceros.

Funciones creadas para generar distintas ondas:

### -Seno

Para crear el seno de una duración y amplitud dada, simplemente se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último se crea con la función sin(), el seno con amplitud y frecuencia frec.

function y = genSeno(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*sin(2\*pi\*frec\*t);

end

### -Coseno

Para crear el coseno es igual que antes, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último se crea con la función cos(), el coseno con amplitud y frecuencia frec.

function y = genCoseno(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*cos(2\*pi\*frec\*t);

end

### -Cuadrada

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último, se crea con la función square(), la señal cuadrada con amplitud y frecuencia frec.

function y = genCuadrada(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*square(2\*pi\*frec\*t);

end

### -Triangular

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último, se crea con la función sawtooth (), la señal triangular con amplitud y frecuencia frec, además, hay que indicar con el argumento ½ que el máximo se encuentra en la mitad del período.

function y = genTriangular(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*sawtooth(2\*pi\*frec\*t,1/2); %Con diente de sierra indicando maximo en la mitad es triangular

end

### -Dientes de sierra

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último, se crea con la función sawtooth (), la señal de dientes de sierra con amplitud y frecuencia frec.

function y = genSierra(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*sawtooth(2\*pi\*frec\*t);

end

### -Chirp lineal desde 0

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último, se crea con la función chirp(t,0,duracion,frec), la señal chirpeada con amplitud y frecuencia que empieza en 0 y aumenta linealmente hasta la frecuencia frec en último instante de tiempo.

function y = genChirpLineal(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

% Creación señal chirp de 0 Hz a frec Hz en t

y=amplitud\*chirp(t,0,duracion,frec); % Empieza @ 0Hz, pasa a frec Hz at t= duracionsec

end

### -Chirp cuadrático

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de -duracion/2 a duracion/2, el vector de tiempos en columnas con el numMuestras calculado. Por último, se crea con la función chirp(t,0,duracion/2,frec,'q'), la señal chirpeada con amplitud y frecuencia que empieza en frecuencia frec y disminuye cuadráticamente hasta la mitad de duración y vuelve a subir hasta la frecuencia frec en último instante de tiempo.

function y = genChirpCuadratico(~,amplitud,frec,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(-duracion/2,duracion/2,numMuestras)';

% Creación señal chirp cuadrático

y=amplitud\*chirp(t,0,duracion/2,frec,'q'); % Empieza @ frec Hz, pasa a frec Hz at t= duracion/2 sec

end

### -Constante

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con esto ya tienes el número de muestras que van a tener el valor de amplitud durante el tiempo deseado.

function y = genConstante(~,amplitud,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Creación señal constante

y=amplitud\*ones(numMuestras,1);

end

### -Lineal

Para el vector de tiempos se vuelve a hacer que, de una duración y amplitud dada, se calcula el número de muestras según la duración en segundos y la Fs en Hz, con linspace se crea de 0 a duracion, el vector de tiempos en columnas con el numMuestras calculado. Por último, solo queda multiplicar la amplitud por ese t para conseguir una recta.

function y = genLineal(~,amplitud,duracion,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

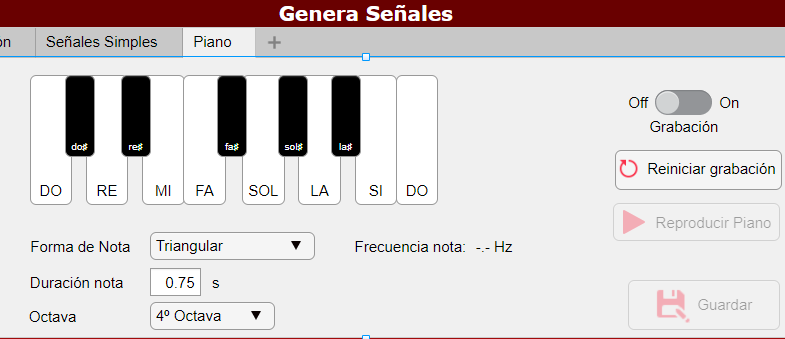
t=linspace(0,duracion,numMuestras)';

% Creación señal

y=amplitud\*t;

end

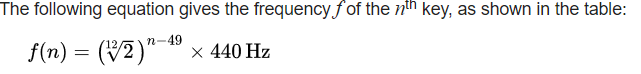
## Piano

Para el piano la función clave es genNota(), que es invocada por cada callback de cada nota del piano: 

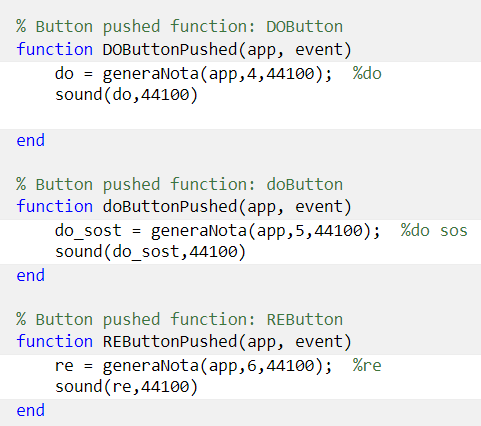
Para entender las notas de un piano, en el piano las siete notas están en orden, comenzando en la nota "DO" y finalizando en "SI", cuando llegamos a la última nota de la escala que es la nota "SI" comienza la escala nuevamente en la siguiente tecla la nota "DO" y se cierra un grupo de teclas al que se lo conoce popularmente con el nombre de octava. En los pianos generalmente hay 7 octavas, en los teclados estándar (no profesionales) hay 5 octavas [3].



Para generar las 108 teclas se usan los “números clave” que van del 1 al 108 y permiten generar las 108 frecuencias fundamentales [4], de cada nota del piano siguiendo la fórmula:



De esta forma cada botón de las 12 notas va a mandar a genNota() un número entero del 4 al 16 y la Fs=44100, que es la clave para calcular a la frecuencia a la que va la nota. De esta forma:



La función:

function nota = generaNota(app,tecla,Fs)

% Número de muestras necesarias a esa Fs y esa duracion

duracion=app.DuracinnotaEditField.Value;

numMuestras=duracion\*Fs;

% Vector de tiempos, en columna

t=linspace(0,duracion,numMuestras)';

%Elección de tecla segun octava

octava=app.OctavaDropDown.Value;

switch octava

case '1º Octava'

clave=tecla;

case '2º Octava'

clave=tecla+12;

case '3º Octava'

clave=tecla+12\*2;

case '4º Octava'

clave=tecla+12\*3;

case '5º Octava'

clave=tecla+12\*4;

case '6º Octava'

clave=tecla+12\*5;

case '7º Octava'

clave=tecla+12\*6;

end

% Según la clave te da la frecuencia con la formula de https://en.wikipedia.org/wiki/Piano\_key\_frequencies

frec = 440\*2^((clave-49)/12);

%Actualiza caja de frecuencia

app.Caja\_frecuenciaLabel.Text=[num2str(round(frec,2)),' Hz'];

%Elección de la forma

forma=app.FormadeNotaDropDown.Value;

switch forma

case 'Triangular'

mid = (t(1)+t(end))/2;

tri = -(abs(t-mid)-mid);

tri = tri./max(tri);

nota = (sin(2\*pi\*frec\*t)).\*tri;

case 'Exponencial'

tt=linspace(duracion,0,numMuestras)';

nota = (sin(2\*pi\*frec\*t)).\*exp(tt);

nota = nota./max(nota);

case 'Seno'

nota = sin(2\*pi\*frec\*t);

case 'Cuadrada'

nota = square(2\*pi\*frec\*t);

case 'Dientes de sierra'

nota = sawtooth(2\*pi\*frec\*t);

end

graficaAudioTiempoFrecuencia(app,nota,Fs,'de última nota del piano')

%plot(app.UIAxesTiempo, (0:length(nota)-1)\*(1/Fs), nota)

%Si el switch de grabar está en On añado la nota a la canción

%con el resto de notas si hubiese

if strcmp(app.GrabacinSwitch.Value,'On')

app.yGrabacionPiano=[app.yGrabacionPiano; nota]; %Concateno la columna de nota

% Ya se puede guardar, reiniciar y reproducir

app.GuardarButton.Enable='On';

app.ReproducirPianoButton.Enable='On';

end

end

Primero calcula el vector de tiempos, de la forma que se ha comentado con el resto de las señales, después según la octava se modifica si es necesario la clave de cada nota, de esta forma puedes tocar cualquier nota de las 108. Con la clave se calcula la frecuencia de la nota, y con la forma introducida por el usuario se crea el seno con envolvente triangular, con caída exponencial u otro tipo de onda. Por último, se muestra la nota y si está la opción de grabar, se guardan las notas por orden de aparición.

## Ruido

Para hacer los distintos tipos de ruido se hace uso de la función cn=dsp.ColoredNoise() que permite crear ruido con las características deseadas. La forma de crear se parece mucho a la de generación de señales simples ya que se parte de los callback de los botones de añadir y reiniciar. Añadir usa una serie de funciones que crean las señales de ruido especificadas por las opciones que hay para introducir por el usuario. Mientras que reiniciar borra la señal creada hasta el momento. Y se reproduce y/o guarda de igual forma. Los distintos ruidos se hacen con el número de muestras que encajen con la duración y Fs deseada. De tal forma que los ruidos, rosa, blanco, marrón, azul y violeta se crean así:

function AadirButtonRuidoPushed(app, event)

%Crea la forma de ruido según lo especificado

color=app.ColordeRuidoDropDown.Value;

Fs=str2double(app.FrecdemuestreoRuidoDropDown.Value);

duracionR=app.DuracionRuidoEditField.Value;

nf = Fs\*duracionR;

switch color

case 'Rosa'

cn=dsp.ColoredNoise('pink','BoundedOutput',true,'SamplesPerFrame',round(nf),'NumChannels',1);

y = cn();

case 'Blanco'

cn=dsp.ColoredNoise('white','BoundedOutput',true,'SamplesPerFrame',round(nf),'NumChannels',1);

y = cn();

case 'Marron'

cn=dsp.ColoredNoise('brown','BoundedOutput',true,'SamplesPerFrame',round(nf), 'NumChannels',1);

y = cn();

case 'Azul'

cn=dsp.ColoredNoise('blue','BoundedOutput',true,'SamplesPerFrame',round(nf),'NumChannels',1);

y = cn();

case 'Violeta'

cn=dsp.ColoredNoise('purple','BoundedOutput',true,'SamplesPerFrame',round(nf),'NumChannels',1);

y = cn();

end

%normalise

maxyb = max(abs(y));

y = y/maxyb;

% Concateno la nueva o primera señal sencilla

app.ySenalRuido=[app.ySenalRuido;y];

app.fsSenalRuido=Fs;

% Muestra la señal creada hasta el momento

graficaAudioTiempoFrecuencia(app,app.ySenalRuido,Fs,'Ruido');

%Una vez que añades señal frecuencia de muestreo no se puede

%volver a tocar

app.FrecdemuestreoRuidoDropDown.Enable='off';

%Se puede reproducir ahora

app.ReproducirRuidoButton.Enable='on';

% Se puede guardar ahora

app.GuardarRuidoButton.Enable='on';

end

Hay que tener en cuenta que para hacer los ruidos de la misma amplitud se deben normalizar entre 1 y -1, para su comparación.

# Cargar/Guardar Archivo

## Cargar Archivo

Para cargar el archivo en el callback de cargar archivo, se usa la función uigetfile que abre una interfaz para poder seleccionar el archivo que se desea abrir, permite abrir las extensiones \*.wav;\*.flac;\*.ogg;\*.mp3;\*.mp4;\*.m4a :

% Abre menú para elegir archivo

[file,path] = uigetfile({'\*.wav;\*.flac;\*.ogg;\*.mp3;\*.mp4;\*.m4a'...

,'Extensiones Audio (\*.wav;\*.flac;\*.ogg;\*.mp3;\*.mp4;\*.m4a)'},'Seleccione el archivo de audio');

if isequal(file,0)

app.FSI011AppSonidoUIFigure.Visible = 'off';

app.FSI011AppSonidoUIFigure.Visible = 'on';

return % Si file es 0, es que ha cancelado y se sale de la función

end

% Hago visible la figura principal de nuevo porque uigetfile te lleva fuera de la ventana

% de mlapp. Con esto se arregla un bug de uigetfile

app.FSI011AppSonidoUIFigure.Visible = 'off';

app.FSI011AppSonidoUIFigure.Visible = 'on';

% También se puede solicionar llamando de nuevo a la figura con figure(app.FSI011AppSonidoUIFigure)

Las líneas de hacer visible e invisible son para solucionar un bug de appdesigner. Para leer el archivo usamos audioread y guardamos los datos en las propiedades:

% Actualizo el valor del archivo seleccionado

app.LocalizacinArchivoLabel.Text=[' ',path];

app.NombreArchivoLabel.Text=file;

% Leo el archivo

[y,Fs] = audioread(fullfile(path,file));

% Guardo el objeto de audio en una propiedad

app.audioSeleccionado = audioplayer(y,Fs);

% Guardo samples en otra convierto si es de 2 canales en 1,

% para procesar. Si es un canal no se modifica y

app.audioSamplesSeleccionado= sum(y, 2) / size(y, 2);

Para el cálculo del ancho de banda usamos obw, que devuelve el ancho de banda dónde se encuentra el 99% de potencia y bandpower() que da la potencia media de la señal.

%calculo de ancho de banda y potencia

[bw,flo,fhi,~] = obw(sum(y, 2) / size(y, 2),Fs);

app.anchoBandaLabel.Text=[num2str(round(bw)),' Hz'];

app.anchoBandaInfLabel.Text=[num2str(round(flo)),' Hz'];

app.anchoBandaSupLabel.Text=[num2str(round(fhi)),' Hz'];

p=bandpower(y);

app.PotenciaMediaVal.Text=[num2str(round(p,6)),' W'];

## Guardar Archivo

Para guardar archivo, por ejemplo, el de la grabación con micrófono es así:

% Button pushed function: GuardarGrabacinButton

function GuardarGrabacinButtonPushed(app, event)

% Abre caja de dialago para decicidir donde guardar y su nombre

% Tipos .wav .mp4 .ogg .flac .m4a, por defecto se llama grabacion.wav

[file,path] = uiputfile({'\*.wav';'\*.mp4';'\*.ogg';'\*.flac';'\*.m4a'},'Guarde la grabación','grabacion.wav');

if isequal(file,0)

app.FSI011AppSonidoUIFigure.Visible = 'off';

app.FSI011AppSonidoUIFigure.Visible = 'on';

return % Si file es 0, es que ha cancelado y se sale de la función

end

% Hago visible la figura principal de nuevo porque uiputfile te lleva fuera de la ventana

% de mlapp. Con esto se arregla un bug de uiputfile

app.FSI011AppSonidoUIFigure.Visible = 'off';

app.FSI011AppSonidoUIFigure.Visible = 'on';

% También se puede solicionar llamando de nuevo a la figura con figure(app.FSI011AppSonidoUIFigure)

% El destino completo es

pathfull=fullfile(path,file);

% Guardamos el archivo

y=getaudiodata(app.grabacionObj); % Audio de la grabacion

Fs=app.grabacionObj.SampleRate; % Muestreo de la grabacion

%bitspersample=app.grabacionObj.BitsPerSample; % Num de bits por muestra de la grabacion

%audiowrite(pathfull,y,Fs,'BitsPerSample',bitspersample,'Comment','Grabacion hecha desde interfaz grafica de Matlab');

% Como no todos los formatos pueden guardar los bits per sample

audiowrite(pathfull,y,Fs,'Comment','Grabacion hecha desde interfaz grafica de Matlab (fsi011)');

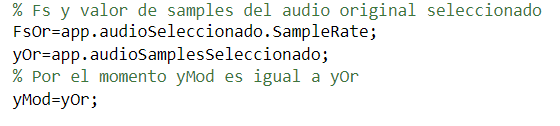
% No todos los archivos soportan las Fs, wav si

end

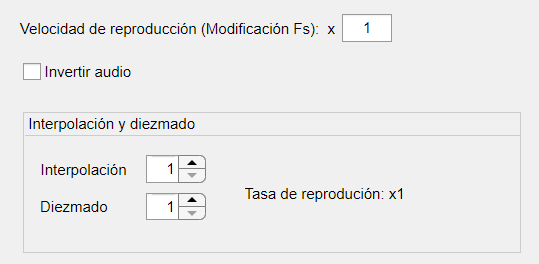
Se usa la función uiputfile para que permita guardar en .wav .mp4 .ogg .flac .m4a, después se obtiene las muestras de sonido “y” y la Fs que se guardan con audiowrite(), añadiendo además un metadato con información sobre la grabación generada.

# Modificar Audio

Todos los cambios se realizan desde el callback del botón realizar cambios. Ahí se extraen las muestras y Fs de la original, y son estás las que se modifican según las opciones del usuario.



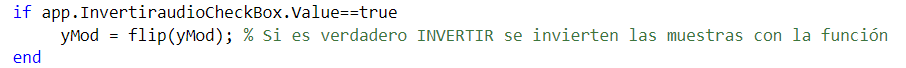
## Velocidad/Inversión



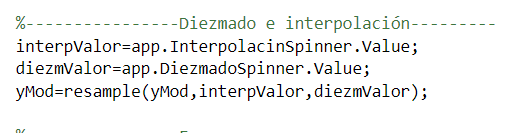
Para la velocidad de reproducción sin perder ni añadir muestras es simplemente multiplicar el valor por la frecuencia de muestreo, para que suene cada muestra más rápido si es mayor que uno, o más lento si es menor que uno:

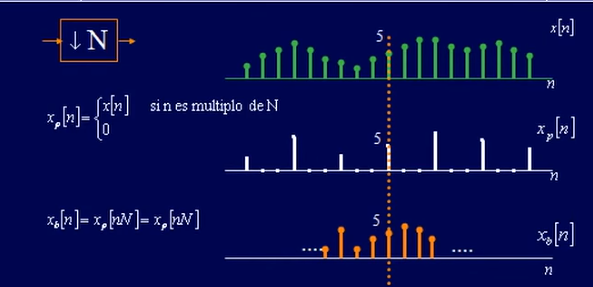


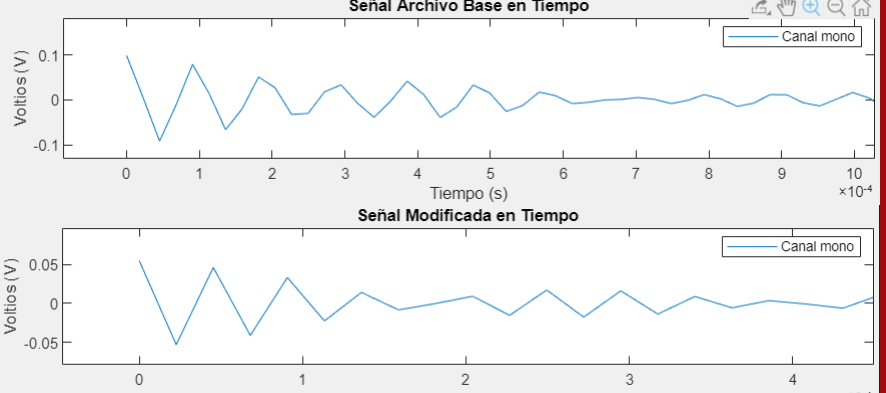
Para invertir las muestras se usa la función flip(), que invierte el orden de las muestras:



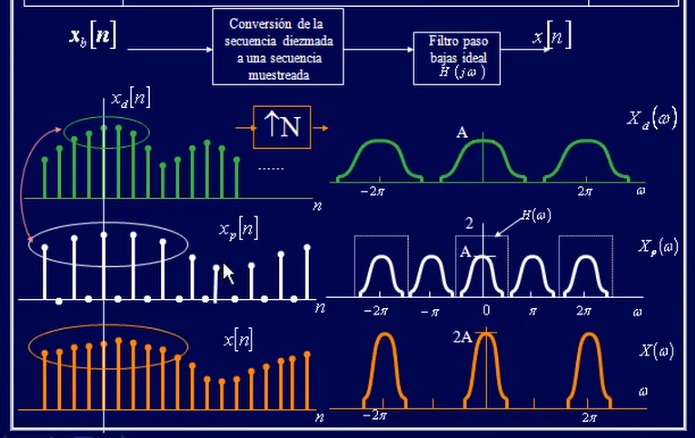
Para diezmar e interpolar, se puede hacer directamente con la función de Matlab resample():



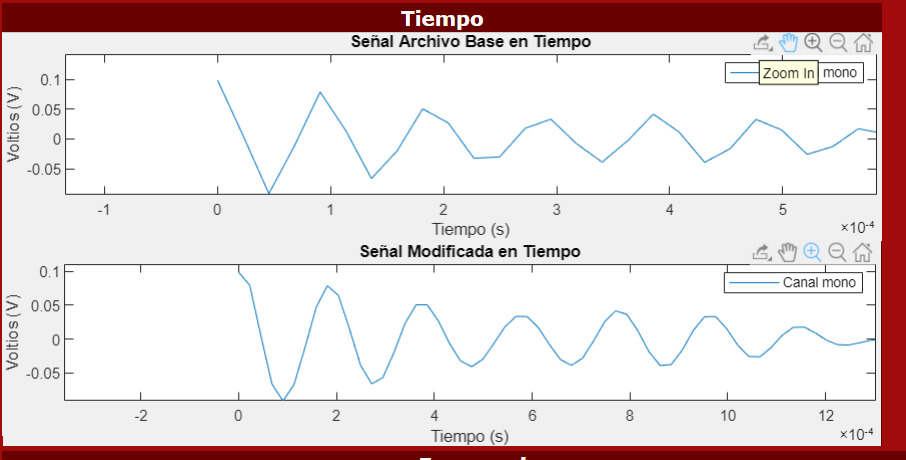
Al diezmar en un factor de 2 significa que te quedas con 1 de cada dos muestras, de 3 con una de cada 3, etc… Creando un efecto de aumento de la frecuencia y por tanto que se reproduzca a mayor velocidad.  Ejemplo con diezmado de 2:



Interpolación añade muestras entre dos muestras con un valor interpolado del resto, si es de valor 2, 1 muestra. Si es de 3, 2 muestras… Crea un descenso de frecuencia, reduciendo la tasa de reproducción.

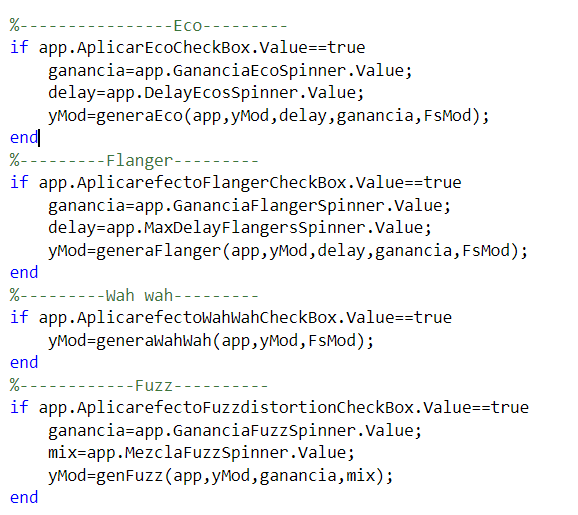


Ejemplo con interpolación de 2:



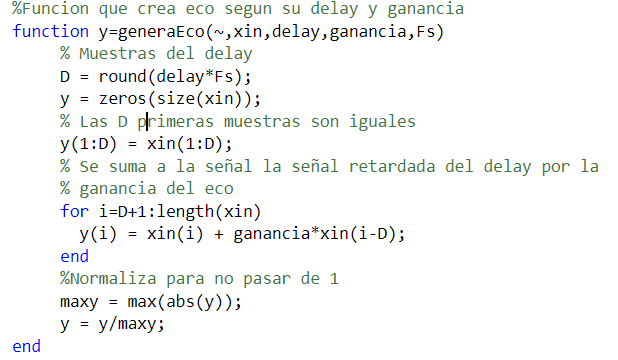
## Efectos

Los distintos efectos se seleccionan en el botón de realizar cambios y por orden son:



### -Eco

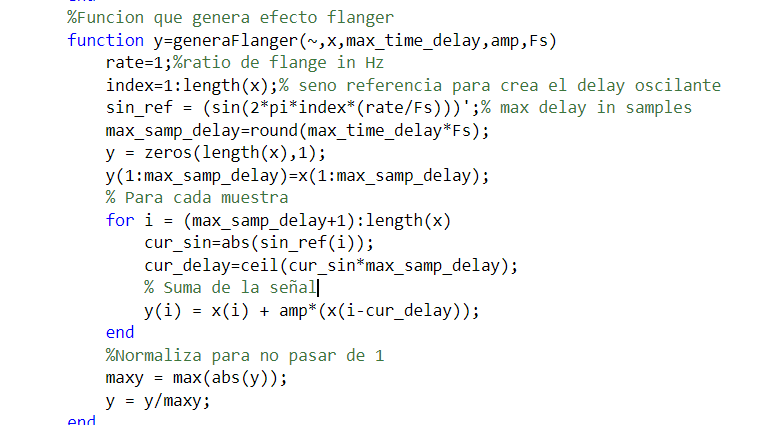
Para añadir el eco partiendo de [5], se calcula el número de muestras según el delay y se van sumando a la señal original una desfasada por el delay multiplicada por una ganancia que es la del eco, después se normaliza, la señal.



### -Flanger

El flanger es un efecto de sonido que produce un característico sonido metalizado oscilante, sobre todo en frecuencias medias y altas. El efecto flanger se obtiene duplicando la onda sonora original; una de las ondas se mantiene limpia de procesado, mientras que a la segunda se le aplica un delay (retraso) menor de 5 milisegundos, con lo que se crea un efecto de comb filter (filtro de peine), que actúa respetando los armónicos. El efecto flanger tiene sus orígenes en las grabaciones magnetofónicas en cinta de bobina abierta. Consiste en mezclar la señal original con una copia retardada en el tiempo, con la particularidad de que el retardo es muy breve, pero varía de forma periódica [6].

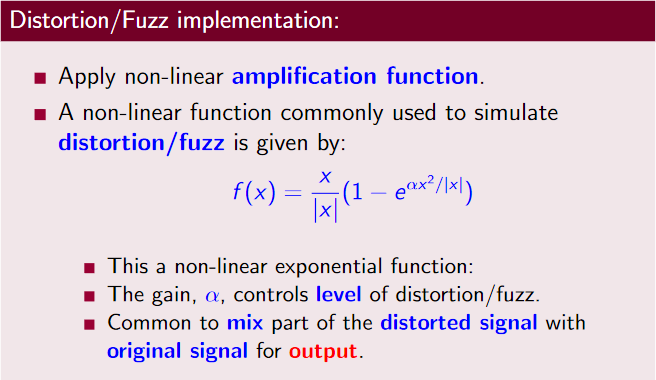
La función que lo genera es function y=generaFlanger(~,x,max\_time\_delay,amp,Fs), al que se le introduce el max tiempo de delay a la señal que se le suma, así como la amplitud de esta, función adaptada de [7].



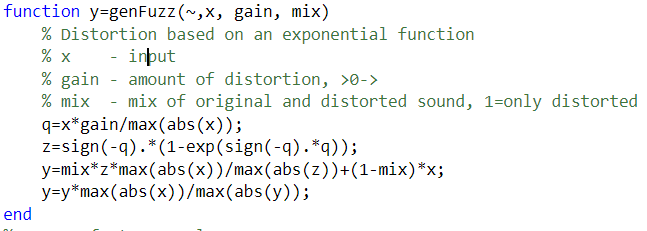
### -Wah-wah

Un filtro paso banda con una frecuencia central modulada variante y con un ancho de banda estrecho. Señal filtrada mezclada con señal directa. Se usa function yb=generaWahWah(~,x,Fs) adaptada de [7].

### -Fuzz

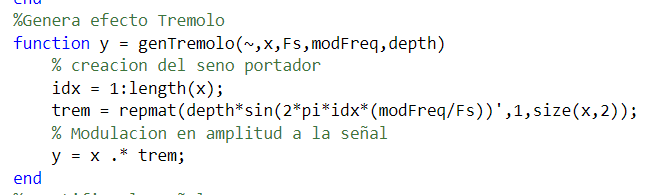
Este efecto se basa en [7]: 

La función genFuzz() se le introduce la ganancia alfa y el mezclado que se quiera hacer, siguiendo la fórmula llegas a:

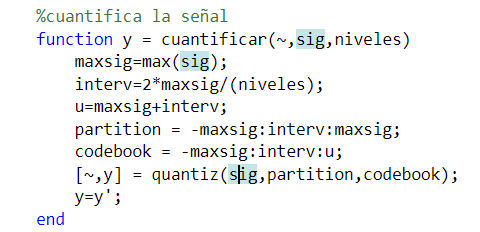


### -Trémolo

Efecto que describe la fluctuación o variación periódica en la intensidad (volumen o amplitud) de un sonido, mientras que la altura o frecuencia se mantiene constante. Función genTremolo al que se indica la profundidad del seno y su frecuencia a la que se varía la amplitud de la señal de audio.



## Cuantificación

Para cuantificar se hace uso de la función quantiz() de Matlab con: 

Se cuantifica la señal en los niveles que se introduzcan. Es necesario trasponer la salida porque devuelve en vector fila.

## Aplicar ruido

Para aplicar ruido a una señal según su SNR, basta con:

%-------------Ruido Gausiano--------

if app.AplicarRuidoBlancoGaussianoCheckBox.Value == true

SNR = app.ValordeSNREditField.Value;

yMod = awgn(yMod,SNR,'measured');

end

agwn(), añade ruido blanco gaussiano a la señal respetando el SNR introducido. Con el atributo ‘measured’ para que tome el verdadero valor de potencia de la señal introducida.

## Filtros

Para la creación de filtros todos se hacen con la función designfilt():

%-------------Filtro--------

if app.AplicarFiltroCheckBox.Value==true

filtro=app.TipodeFiltroDropDown.Value;

frecCorte=app.FrecuenciadecorteHzEditField.Value;

frecBanda1=app.FrecuenciaPaso1HzEditField.Value;

frecBanda2=app.FrecuenciaPaso2HzEditField.Value;

switch filtro

case 'Paso bajo'

f = designfilt('lowpassfir','PassbandFrequency',frecCorte, ...

'StopbandFrequency',frecCorte+20,'DesignMethod','kaiserwin', ...

'SampleRate', FsMod);

case 'Paso alto'

f = designfilt('highpassfir','PassbandFrequency',frecCorte, ...

'StopbandFrequency',frecCorte-20,'DesignMethod','kaiserwin', ...

'SampleRate', FsMod);

case 'Paso banda'

f = designfilt('bandpassfir', 'StopbandFrequency1', frecBanda1-20, ...

'PassbandFrequency1', frecBanda1, 'PassbandFrequency2', frecBanda2, ...

'StopbandFrequency2', frecBanda2+20,'DesignMethod','kaiserwin', ...

'SampleRate', FsMod);

case 'Elimina banda'

f = designfilt('bandstopfir', 'StopbandFrequency1', frecBanda1+20, ...

'PassbandFrequency1', frecBanda1, 'PassbandFrequency2', frecBanda2, ...

'StopbandFrequency2', frecBanda2-20,'DesignMethod','kaiserwin', ...

'SampleRate', FsMod);

end

%Se filtra la señal con el filtro pedido

yMod = filter(f,yMod);

%Se vea el filtro

%..................FTT.................

%freqz calcula respuesta del filtro de 10000 puntos

[Y\_espectroPos,frec]=f.freqz(10000,FsMod);

plot(app.UIAxesFiltro,frec,20\*log10(abs(Y\_espectroPos)));

app.UIAxesFiltro.YLimMode='auto';

value = app.VerhastafrecuenciaHzEditField\_3.Value;

% Permite escalar el eje x hasta la frecuencia deseada

app.UIAxesFiltro.XLim=[0 value];

end

Todos los filtros se hacen con ventanas de kaiser y sus frecuencias de corte se indican por el usuario mientras que las frecuencias de parada se calculan 20 Hz por encima o por debajo de la de corte, según el tipo de filtro. Para visualizar el filtro se usa la función freqz(), al que se indica el número de puntos y la Fs.

Para ver toda la información del filtro se usa info(), y a través del callback muestra todas las características con:

%Obtiene info del filtro

s=info(f);

%Para representar cada campo

n=repmat('\n',size(s,1),1);

s2=[s,n];

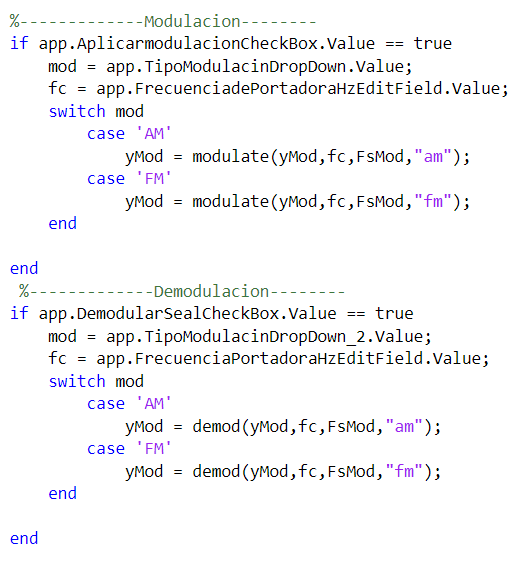
B = convertCharsToStrings(s2');

%Crea una ventana de dialogo con el mensaje

uialert(app.FSI011AppSonidoUIFigure,sprintf(B),'Información de filtro','Icon','info');

## Modulación/Demodulación

La modulación y demodulación AM y FM se pueden hacer fácilmente con las funciones modulate() y demod(). La frecuencia de la portadora se introduce por parte del usuario, pero ésta debe ser menor que la mitad de la frecuencia de muestreo por el teorema de Nyquist. Por tanto, el código para la modulación y la demodulación es el siguiente:



# Referencias

[1] MATLAB App - Add path before component creation. (2018, 5 julio). Recuperado 5 diciembre, 2019, de https://stackoverflow.com/questions/51189516/matlab-app-add-path-before-component-creation

[2] J.G. Proakis y M. Salehi, Contemporary communication systems using Matlab, Ed.

Brooks/Cole, 2000.

[3] Lección 3: Las notas del piano - EstudiarPiano.com. (s.f.). Recuperado 6 diciembre, 2019, de http://www.estudiarpiano.com/aprender-piano/las-notas-del-piano

[4] Wikipedia contributors. (2019, 6 diciembre). Wikimedia list article. Recuperado 6 diciembre, 2019, de https://en.wikipedia.org/wiki/Piano\_key\_frequencies

[5] Ashan Peiris, A. (s.f.). How to add an echo effect to an audio signal using Matlab. Recuperado 6 diciembre, 2019, de http://ashanpeiris.blogspot.com/2015/03/how-to-add-echo-effect-to-audio-signal.html

[6] Colaboradores de Wikipedia. (2019, 7 diciembre). Flanger - Wikipedia, la enciclopedia libre. Recuperado 8 diciembre, 2019, de https://es.wikipedia.org/wiki/Flanger

[7] Marshal, D. (s.f.). CM3106 Chapter 7:Digital Audio Effects. Recuperado de users.cs.cf.ac.uk/Dave.Marshall/Multimedia/PDF/07\_Audio\_Effects.pdf

[8] Rampartisan, A. (s.f.). rampartisan/Matlab\_Audio\_Effects. Recuperado 8 diciembre, 2019, de https://github.com/rampartisan/Matlab\_Audio\_Effects/blob/master/MAE\_Tremolo.m