MÉMORIA PRÁCTICA PDS

FASE 1

Explicación del código

Para desarrollar el ejemplo1 y ejemplo2, hemos tenido que usar la función *parameq.m*, que recibe una serie de parámetros y devuelve el numerador y denominador del filtro global, *B* y *A* respectivamente, y la señal filtrada *y*.

La función *parameq.m* encuentra la salida *y* basándose en el funcionamiento del esquema del filtro proporcionado por el enunciado. Tras realizar los cálculos en papel, se encuentra el valor de N y M, y con estos valores se encuentran B y A.

```
K = 10.^(G/20);
b = -cos((2*pi*f0)/(fs));
a = (1 - tan(2*pi*(bw/fs)))/(1 + tan(2*pi*(bw/fs)));
%Numerador
num = [a b*(1+a) 1];
%Denominador
den = [1 b*(1+a) a];
%Funcion de Transferencia
Hap = num/den;
%Filtro
%Ecuación de la y
y = (1/2)*(x + x*Hap) + (K/2)*(x - x*Hap);
%A partir del filtro aislamos la M y la N
M = (1 + K)/2;
N = (1 - K)/2;
B = [(M+N*a) b*(1+a)*(M +N) (M*a+N)];
A = den;

yFinal = filter(B, A, y);
```

Posteriormente para cada ejemplo hemos usado unos valores diferentes de entrada para el *parameq.m*, pero la idea es la misma en ambos casos.

A partir de los datos que especifica el enunciado, se construye la señal de entrada correspondiente, y posteriormente se obtienen también los datos devueltos por *parameq.m*:

```
G = 30;
fs = 8000;
f0 = 2000;
bw = db2mag(6);

T = 0.1;
n = round(fs*T);
N = 0:(n-1);
x = 3*(cos(2*pi*0/fs*N)) + cos(2*pi*500/fs*N) + 3*(cos(2*pi*1900/fs*N)) + 2*(cos(2*pi*2000/fs*N));
[B,A,y] = parameq(f0,fs,bw,G,x);
```

Se obtiene la señal en el dominio frecuencial mediante la transformada de Fourier y se muestra. Para mostrarla correctamente, se aplica un *20*log10(abs(z))*. Esto se hace en todos los plots de señales en dominio frecuencial de la práctica:

```
z = fft(x);
n = length(x);
z = z(1:n/2+1);
f = 0:fs/n:fs/2;

%Módulo en dB de la señal de entrada en el dominio frecuencial
subplot(3,2,1)
plot(f, 20*log10(abs(z)))
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
title('Input','fontweight','bold')
grid on
xlim([0 4000])
ylim([40 90]);
```

Para mostrar la señal x en el dominio temporal simplemente se muestra tal cual:

```
%Señal de entrada en el dominio temporal
subplot(3,2,2)
plot(x)
xlabel('Sample')
ylabel('Amplitude')
title('Input','fontweight','bold')
grid on
xlim([0 800])
ylim([-2.5 9]);
```

Para mostrar la respuesta del filtro, se usan los valores B y A devueltos por *parameq.m*. En el dominio frecuencial se usa la función *freqz*, y en el dominio temporal es necesario crear un array entero de ceros, excepto un único 1.

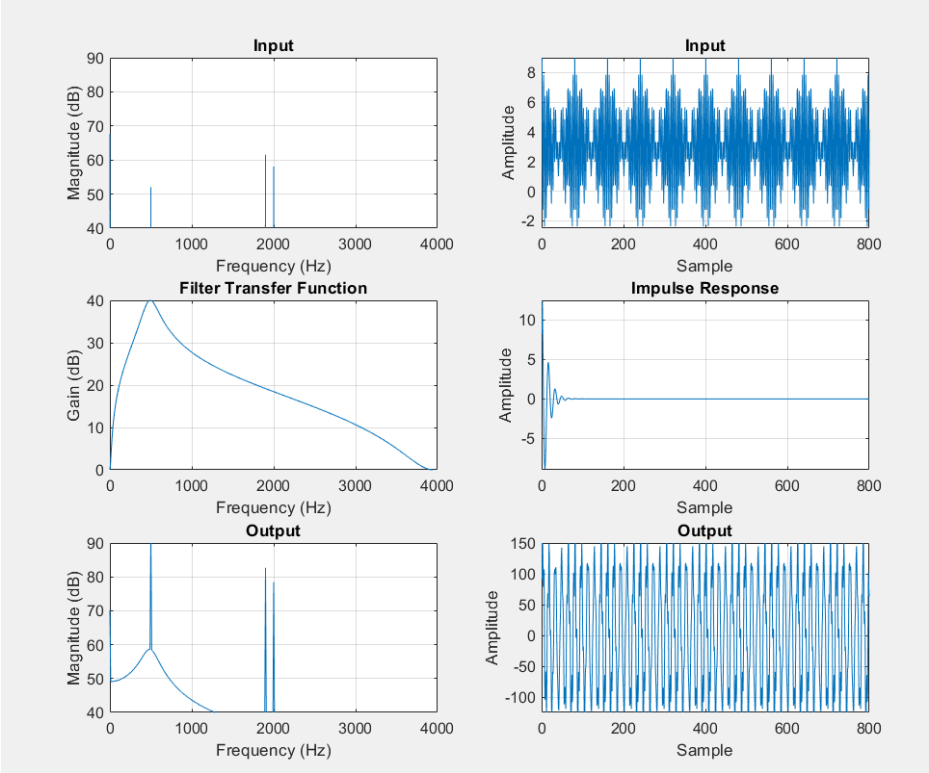
```
%Respuesta del filtro en el dominio frecuencial
subplot(3,2,3)
[h,w] = freqz(B,A);
H = 20*log10(abs(h));
plot(w*(400*pi), H)
xlabel('Frequency (Hz)')
ylabel('Gain (dB)')
title('Filter Transfer Function','fontweight','bold')
grid on
xlim([0 4000])
ylim([0 40]);

%Respuesta del filtro en el dominio temporal
subplot(3,2,4)
arrayZeros = zeros(1, 3999);
H = [1 arrayZeros];
w = filter(B,A,H);
plot(w)
xlabel('Sample')
ylabel('Amplitude')
title('Impulse Response','fontweight','bold')
grid on
xlim([0 800])
ylim([-9 12.5]);
```

Para mostrar la señal filtrada en el dominio frecuencial y temporal, repetimos el proceso que habíamos hecho con la señal de entrada *x*, pero ahora con la señal de salida *y*:

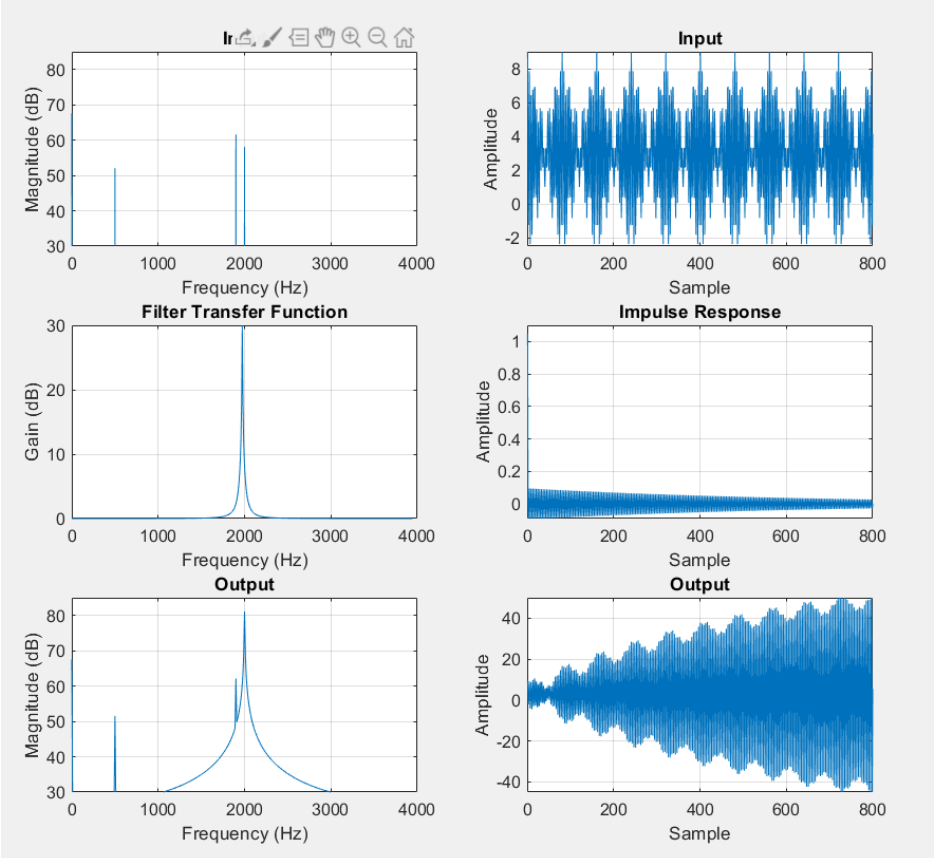
```
%Módulo en dB de la señal filtrada en el dominio frecuencial
yFreq = fft(y);
lengthY = length(y);
yFreq = yFreq(1:lengthY/2+1);
fY = 0:fs/lengthY:fs/2;
yDB = 20*log10(abs(yFreq));
subplot(3,2,5)
plot(fY, yDB)
xlabel('Frequency (Hz)')
ylabel('Magnitude (dB)')
title('Output','fontweight','bold')
grid on
xlim([0 4000]);
ylim([40 90]);

%Señal de salida filtrada en el dominio temporal
subplot(3,2,6)
plot(y);
xlabel('Sample')
ylabel('Amplitude')
title('Output','fontweight','bold')
grid on
xlim([0 800]);
ylim([-125 150]);
```

Resultados obtenidos

En el ejemplo 1:



En el ejemplo 2:



FASE 2

Funcionamiento de la interfaz

Esta segunda fase de la práctica consiste en crear una interfaz gráfica que permita cambiar el ancho de banda, ganancia y frecuencial central de un filtro mediante controles, mostrando gráficamente el resultado.

El filtro usado funciona mediante la función *parameqFullband,* que no es más que la concatenación de dos filtros en cascada, por lo que simplemente se llama dos veces al *parameq* normal, pasándole como input x al segundo *parameq*, el output y devuelto por el primer *parameq*. La B y A finales se consiguen convolucionándolas usando la función *conv* de Matlab.

```
%Función parameq_fullband.m
function [B,A,y_filtrada] = parameq_fullband(f0,fs,bw,G,x)
% Función encargada de implementar el filtro global formado por la
% concatenación en serie o cascada de dos filtros paramétricos (función
% parameq.m)

[B1, A1, y1] = parameq(f0(1),fs,bw(1),G(1),x);
[B2, A2, y2] = parameq(f0(2),fs,bw(2),G(2),y1);
y = y2;

B = conv(B1, B2);
A = conv(A1, A2);

y_filtrada = filter(B,A,y2);
```

Para la interfaz, se han programad varias funciones necesarias para el funcionamiento del programa. Dos de ellas, *SavefileMenuSelected* y *OpenfileMenuSelected,* permiten guardar un audio o abrirlo, respectivamente. Para guardar el audio se usa la función de Matlab *audiowrite,* y para abrirlo la función *uigetfile*:

```
% Menu selected function: SavefileMenu
function SavefileMenuSelected(app, event)
    audiowrite(app.output,app.yOutput,app.fs);
end

% Menu selected function: OpenfileMenu
function OpenfileMenuSelected(app, event)
   try
        app.input= uigetfile('.wav');
        [app.xInput, app.FsInput] = audioread(app.input);
        app.player = audioplayer(app.xInput, app.FsInput);
        actualizaGraficas(app, "");
   catch
        disp("Ha ocurrido un error al abrir el fichero");
        return;
   end
end
```

Para reproducir el audio se ha utilizado la función *audioread,* ya que lee el archivo y obtiene su Fs y la señal en sí, y con eso se crea un *audioplayer*, que permitirá ir haciendo *play* o *pause*.

```
% Button pushed function: PlayButton
function PlayButtonPushed(app, event)

    if(app.Switch_2.Value == "Original")
        [app.xInput,app.FsInput] = audioread(app.input);
        app.player = audioplayer(app.xInput,app.FsInput);
        play(app.player);
    else
        app.player = audioplayer(app.yOutput,app.fs);
        play(app.player);
    end

end

% Button pushed function: StopButton
function StopButtonPushed(app, event)
    stop(app.player);
end
```

Para mantener actualizada la interfaz, cada vez que cambian los valores de algunos de los componentes de la interfaz, se actualiza ese valor que ha sido cambiado (ganancia, ancho de banda o frecuencia central) en la *app*, se llama a la función *parameqFullband* para conseguir la señal de salida actualizada, y finalmente se llama a la función *plotsignals* para que muestre las gráficas actualizadas:

```
% Value changed function: CenterFreqHzKnob
function CenterFreqHzKnobValueChanged(app, event)
    value = app.CenterFreqHzKnob.Value;
    app.f0(1) = value;
    [B, A, app.yParameq] = parameq_fullband(app.f0, app.fs, app.bw, app.GB, app.xInput);
    plotsignals(app, B, A);
    close all
end
```

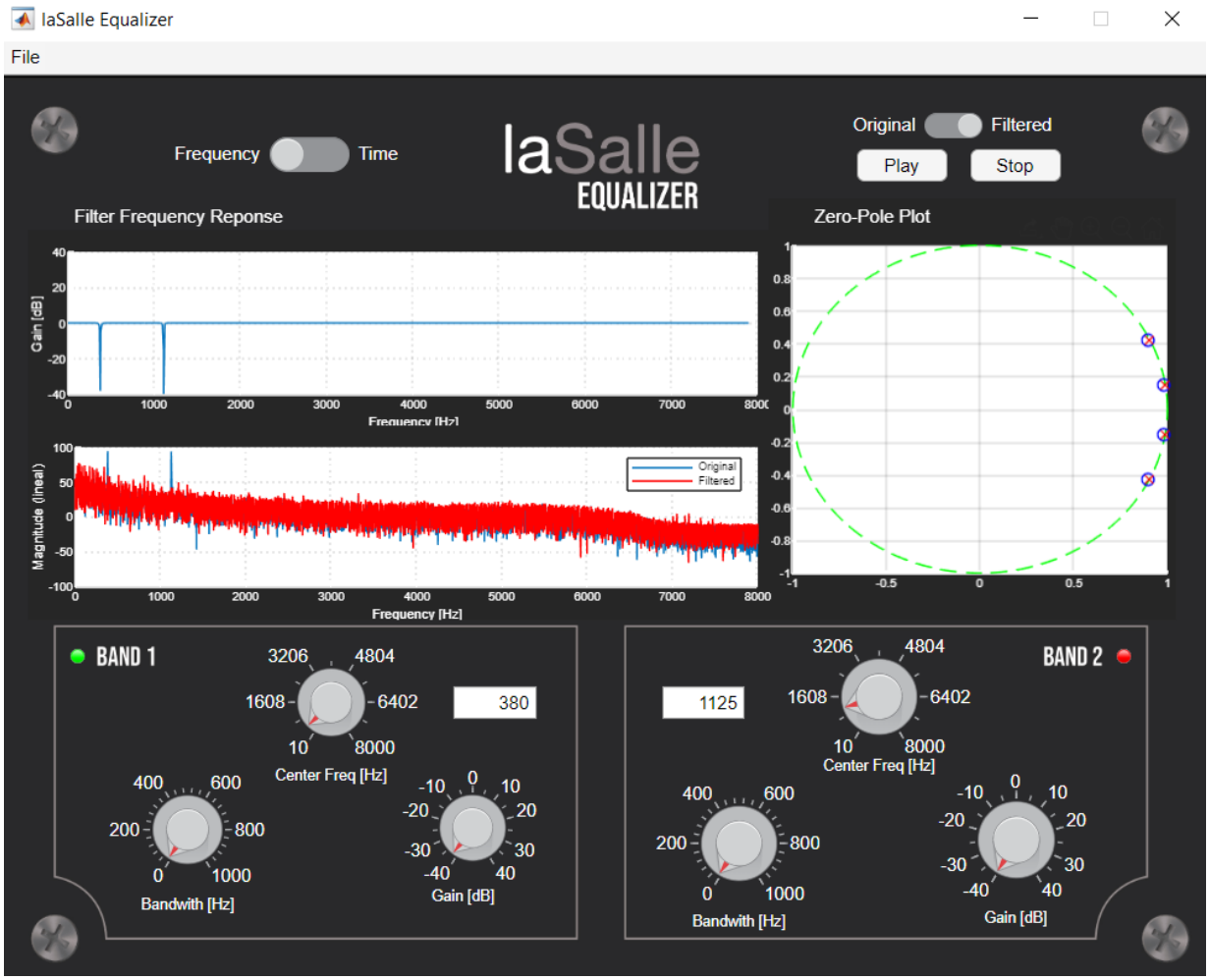
En la función *plotsignals*, se comprueba si se quiere mostrar la señal de salida en dominio temporal o frecuencial. Si es dominio temporal, se muestra directamente la señal de entrada (el audio original) y la salida (la señal que se obtiene de *parameqFullband).* Si es dominio frecuencial, se usa la transformada de Fourier. Para mostrar los ceros y polos, se usa de Matlab *roots* con la A y B que se obtiene de *parameqFullband.*

```
 function plotsignals(app, B, A)
     %Respuesta frecuencial
     [h,w] = freqz(B,A, app.fs);
     H = 20*log10(abs(h));
     plot(app.UIAxes, w*(800*pi), H)
     xlabel(app.UIAxes, 'Frequency [Hz]')
     ylabel(app.UIAxes, 'Gain [dB]')
     ylim(app.UIAxes, [-40 40]);

     %Señal (original y filtrada)
     if(app.Switch.Value == "Time")
         %Original
         plot(app.UIAxes3, app.xInput);
         hold(app.UIAxes3, 'on')

         %Filtrada
         plot(app.UIAxes3, app.yParameq, 'color', [0.8, 0.3, 0.1]);
         hold(app.UIAxes3, 'off');
         xlabel(app.UIAxes3, 'Seconds')
         ylabel(app.UIAxes3, 'Amplitude')
         legend(app.UIAxes3,'Original','Filtered');
     end

if (app.Switch.Value == "Frequency")
    %Original
    x = app.xInput;
    x_freq = fft(x);
    lengthY = length(x_freq);
    x_freq = x_freq(1:lengthY/2+1);
    x_freq = 20*log10(abs(x_freq));
    fx = 0:app.fs/lengthY:app.fs/2;

    plot(app.UIAxes3, fx, x_freq);
    hold(app.UIAxes3, 'on')

    %Filtrada
    y = app.yParameq;
    y_freq = fft(y);
    lengthY = length(y);
    y_freq = y_freq(1:lengthY/2+1);
    y_freq = 20*log10(abs(y_freq));
    fy = 0:app.fs/lengthY:app.fs/2;

    plot(app.UIAxes3, fy, y_freq, 'r');
    hold(app.UIAxes3, 'off');
    xlabel(app.UIAxes3, 'Frequency [Hz]')
    ylabel(app.UIAxes3, 'Magnitude (lineal)')
    legend(app.UIAxes3,'Original','Filtered');
end

   cla(app.UIAxes2);
   %Creamos el círculo unidad
   radio = 1;
   circulo_unidad = 0:pi/30:2*pi;
   x = radio*cos(circulo_unidad);
   y = radio*sin(circulo_unidad);
   plot(app.UIAxes2, x,y,'--g');
   %Con roots obtenemos las raíces de los polinomios.
   ceros=roots(A);
   polos = roots(B);
   plot(app.UIAxes2,ceros,'ob');
   hold(app.UIAxes2, 'on');
   plot(app.UIAxes2,polos,'xr');
   xlim(app.UIAxes2, [-1 1]);
   ylim(app.UIAxes2, [-1 1]);
   close all
end
```

Ejercicio interferencia

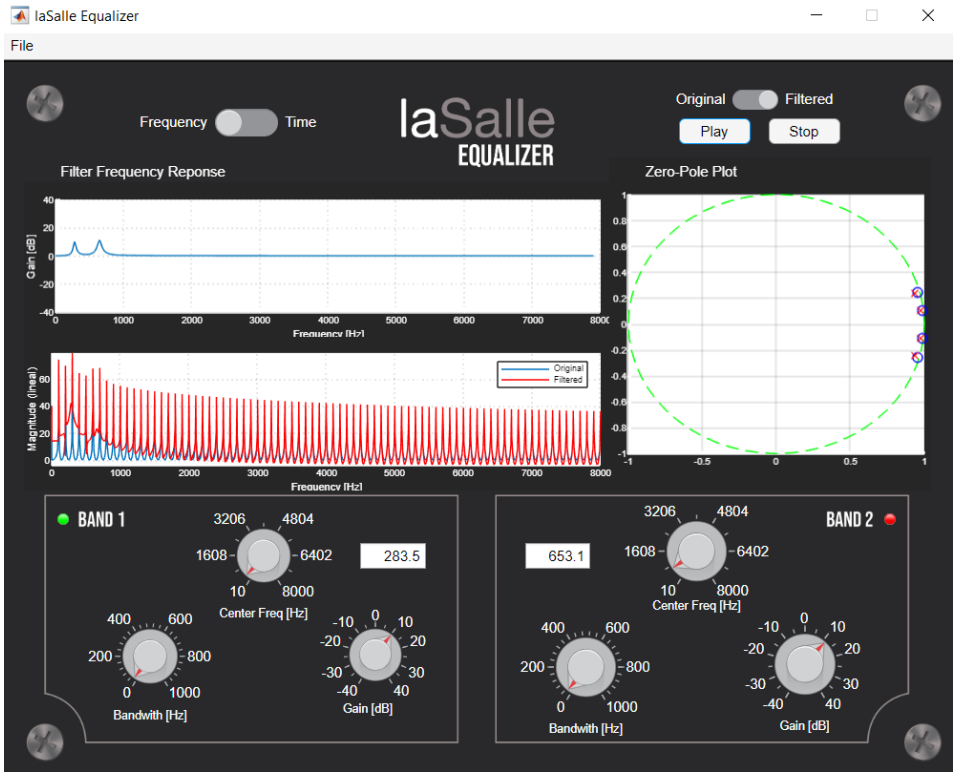
Para encontrar las frecuencias donde se produce el ruido, hay que observar la señal de entrada en el dominio frecuencial. Se pueden apreciar claramente dos picos muy diferenciados que sobresalen del resto de la señal. Si se hace zoom en la gráfica y se marcan esos dos puntos, se ve que se encuentran en las frecuencias 380 y 1125.

Así pues, para eliminar el error, se sitúa la frecuencia central del primer filtro en 380 y la del segundo filtro en 1125. Se pone un ancho de banda muy pequeño en ambos filtros, ya que el ruido se produce en unas frecuencias muy concretas y marcadas, así que hay que ser lo más precisos posibles para modificar solo esas frecuencias y no otras, ya eso alteraría la música. Por último, se la ganancia en negativo en ambos filtros. Con ello se obtiene un resultado sin ruido y con la música lo menos modificada posible.



Ejercicio vocales

Para este ejercicio, se han encontrado los diferentes sonidos vocales a partir de las frecuencias centrales proporcionadas por la tabla del enunciado, con un ancho de banda muy bajo y una ganancia entorno a los 10 dB. Abajo se muestra el ejemplo de obtención del sonido vocal *u*.



Posteriormente, mediante el uso de un script, se consiguen unir todos los sonidos vocales y se obtiene el fichero '.wav'.

concatenaVocales.m

```
%Funcion para generar las vocales concatenadas
function concatenaVocales
[x_a, fs_a] = audioread('a.wav');
[x_e, fs_e] = audioread('e.wav');
[x_i, fs_i] = audioread('i.wav');
[x_o, fs_o] = audioread('o.wav');
[x_u, fs_u] = audioread('u.wav');
x_final = x_a;
x_final = [x_final;x_e];
x_final = [x_final;x_i];
x_final = [x_final;x_o];
x_final = [x_final;x_u];

audiowrite('aeiou.wav', x_final, fs_a);

```

CONCLUSIONES Y PROPUESTAS DE MEJORA

Finalmente, como conclusión, me gustaría comentar algunos de los problemas que tuve al realizar la práctica.

El problema principal lo tuvimos en la entrega de enero con la fase 2, y es que el ecualizador no funcionaba correctamente. Para cargar las gráficas de las señales tardaba varios minutos, incluso llegaba a la media hora a veces. Esto hacía imposible poder hacer pruebas con los valores que introducíamos. Además, por falta de tiempo, no conseguimos mostrar la gráfica de ceros y polos ni realizar los ejercicios de interferencia y vocales.

En esta reentrega, por el contrario, he podido dedicarle mucho más tiempo. Pude resolver el problema de rendimiento eliminando una función que creamos y que se llamaba antes que *plotsignals,* pude incorporar la gráfica de polos y ceros, y también realicé los dos ejercicios propuestos.