Assignment 11: Multirate Signal Processing

Colt Thomas

January 25, 2021

1 Problems

Problem 11.1

Using MATLAB construct a sampled version of the analog signal

$$x(t) = \sum_{r=1}^{8} A_m sin(2\pi F_r t)$$

Creating N=2048 samples,xn=x(n) for n=[0:2047], at sample rate of 4000 samples/sec. The Am and Fm values are defined in the following MATLAB command sequence:

```
N=2048; n=0:N-1; nseg=1000:1060;
Hz=1;KHz=1e3; Fs=4000*Hz; Ts=1/Fs;

F=[100:50:450].'; A1=[1:4]/4; A=[A1,fliplr(A1)];
xn=A*sin(2*pi*F*n*Ts);
```

This will create a band limited consisting of 8 sinusoidal tones with linear increasing and decreasing amplitudes. With full MATLAB documentation (with the exception of mfiles provided as part of this course), the fiveproblem requirements (A through E) are:

Part A

Display with appropriate title and axis labels the sampled signal x(n) as a function to sample time (ms) over the full time range equating to N samples

See part A of the matlab code, and note how we create the time vector in miliseconds on line 15. Also see the top plot in Figure 1 for the analog representation of the signal described above.

Part B

Display with appropriate title and axis labels the sampled signal x(n) as a function to sample time (ms) for a segment of the signal with indexes nseg=[1000:1060].

For this part, all we have to do is access the index range 1000:1160 of our analog signal x. We are essentially zooming in on our signal, and viewing a sampled representation of the signal (hence the stem plot). See plot 2 of Figure 1.

Part C

Compute and display with appropriate title and axis labels the magnitude of dtft(xn,n,w)/N for ω/π *going from -2 to 2.*

The spectrum of $x(n)$ was created using the class-given dtft() function. See the Matlab code below, and the plot 3 of Figure 1.

Part D

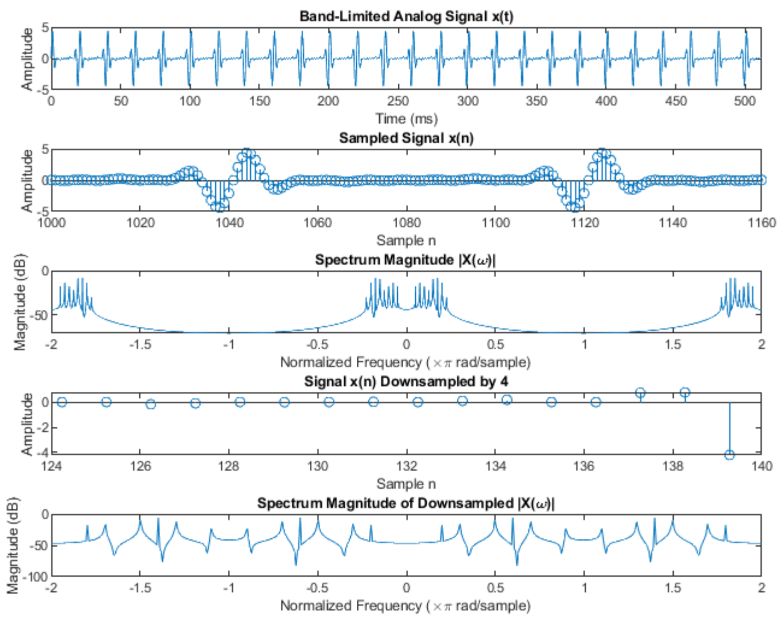
Using the MATLAB built in "downsample' function reduce the sample rate of x(n) by 4, i.e., y(m)=x(4m) where $0 \leq m \leq \frac{N}{4} - 1$ *Display with appropriate title and axis labels for the reduced sampled signal y(m)a function reduced sample times (ms) i.e., for m=[125:140].*

Plot 4 shows the reduced sampled signal for index range 125:140 in Figure 1. Notice that the samples don't quite line up on the millisecond marks on the x axis. This is because our linspace function creates evenly spaced values from 0 to the full duration of the signal.

Part E

Compute and display the magnitude dtft(y,m,w)/N with appropriate labeling for ω/π *going from -2 to 2.*

Whenever a signal is downsampled, it should also be passed through a low-pass filter. In our case of downsampling D=4, we reduced our "sample rate" from 4000Hz to 1000Hz. The nyquist frequency of our new signal is now 500Hz. In the last plot of Figure 1, we don't necessarily have aliasing going on (highest frequency is 450Hz), but we do see that the spectrum spread out. This is because downsampling reduces bandwidth of the signal.



Band-Limited Analog Signal x(t)
Amplitude
Time (ms)
Sampled Signal x(n)
Amplitude
Sample n
Spectrum Magnitude |X(ω)|
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Signal x(n) Downsampled by 4
Amplitude
Sample n
Spectrum Magnitude of Downsampled |X(ω)|
Magnitude (dB)
Normalized Frequency (×π rad/sample)

Problem 11.2

In problem 11.1 two discrete time signals were created by, first sampling the analog signal x(t) at a rate to 4000 and secondly downsampling x(n) by a factor of 4 to get y(m).Fully document your MATLAB script files (excluding m files provided to you as part of this course)

Part A

Using MATLAB reconstruct x(t) from x(n) using the sinc interpolation technique. Plot the recovered signal for the time region used for signal display purposed in the previous problem (11.1) with an interpolation segment of $\Delta t = 0.05$ *ms. This is a sufficiently fine interpolation increment that we will assume approximates an underlying continuous-time (analog) signal.*

For comparison, I used the interp() command and plotted the results of interpolating x(n) by 4 in the first plot of 1. Notice that there is a substantial difference between the interpolated signal and the reconstructed signal.

Part B

Repeat A using y(m) and compare and discuss the results of A and B.

Reconstructing our downsampled signal y(n) yields the original signal x(t). Closer examination shows that one waveform of the x(n) reconstruction spans the same amount of time for three waveforms in the reconstruction of y(n). It's as if the reconstruction of x(n) somehow stretched out the original signal.

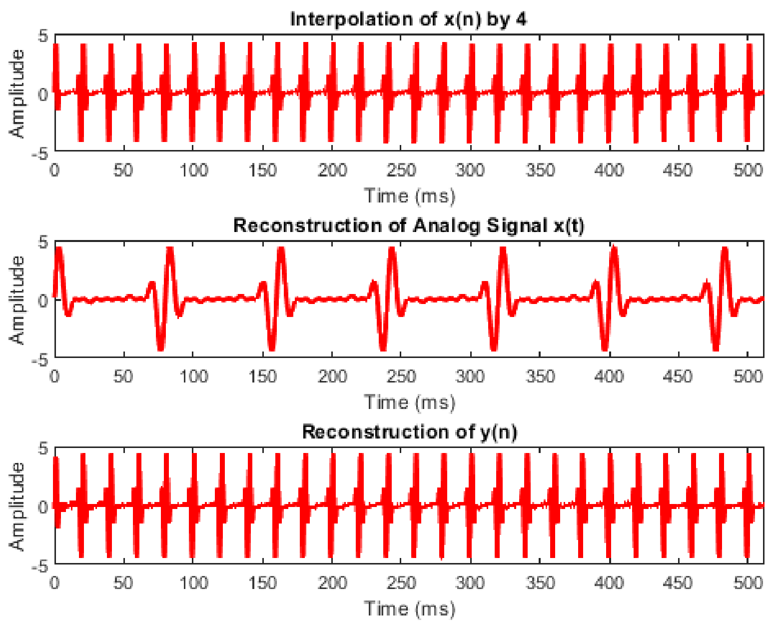


Figure 1: Reconstruction of the original signal x(n) vs the downsampled signal y(n)

Problem 11.3

Again full MATLAB documentation is required as in the previous problems. Here students are to

Part A

Display a segment of the new signal x(n)

See plot 2 of Figure 2

Part B

Calculate and display the dtft of x(n)

See plot 3 of Figure 2

Part C

Upsample the signal to y(m)=x(n/4) where x is an expanded signal that is a zero except for indexes n equal to a multiple of 4. Plot x using the sampletimes in part A.

See plot 4 of Figure 2

Part D

Compute the dtft of the expanded signal x(n/4)and comment of the results. radian frequency that includes –pi to +pi.

See plot 5 of Figure 2

Part E

Design and import to the MATLAB workspace an appropriate LP filter to recover the upsampled signal. Compare the signals x(n) and filtered y(m).

When we upsample our signal, the DTFT will of the upsampled signal will produce spectral copies at $\frac{\pi}{I}$, where I is the upsample value. To remove the extra spectral copies, we can pass our signal through a lowpass filter after we upsample it. Below in Figure 3 is a screenshot of the Filter Designer window with the specifications.

Problem 11.4

In this problem we will consider the transmission of data consisting of numbers from 0 to 9. The sending signal will consist to two tones with frequency pairs associated with the data values. This system is known as the dual tone multiple

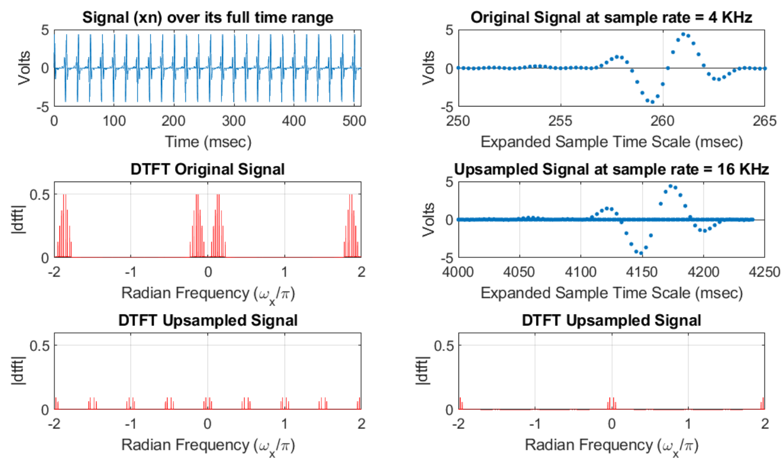


Figure 2: Plots showing the process and effects of upsampling a signal.

frequency (dtfm) system. It is commonly used by key pads to send controlsignals over an audio link. This is illustrated below.

When a specific key is pushed then then two sinewave tones are transmitted based on the row and column locations of the key. This tone system is simulated by the MATLAB function $[x, Fs] = dtmf(phoneNum)$

*The input variable phoneNum is a vector array of integers for -2 to 9 where -1 equates to the * key and -2 equates to the # key. We'll only use the the 0 to 9 keys for this problem. For this problem phoneNum will be a row vector of ten single digit numbers from 0 to 9. Experiment with* $phoneNum = [1234567890]$

When you examine the function you will see the sample rate, Fs, the length of the tone, T, and the guard time (dead time between successive tones), tau. The function creates a signal x(n) consisting of multiple dual tone segments that is then transmitted to a receiver. The command $soundsc(x, Fs)$ *sends the signal x and sample rate to your speaker. When you run the function you will recognize the touch tone sounds that you make hear from time to time. You can comment the command out when the novelty of hearing the tones wears off.*

The objective of this problem is to design a combination of filters to detect the tones, represent the power associated with the output of the filter, to sample the power output of the filter and determine the transmitted datasequence (integers).

You will need to design 7 pass band filters with center frequencies matching

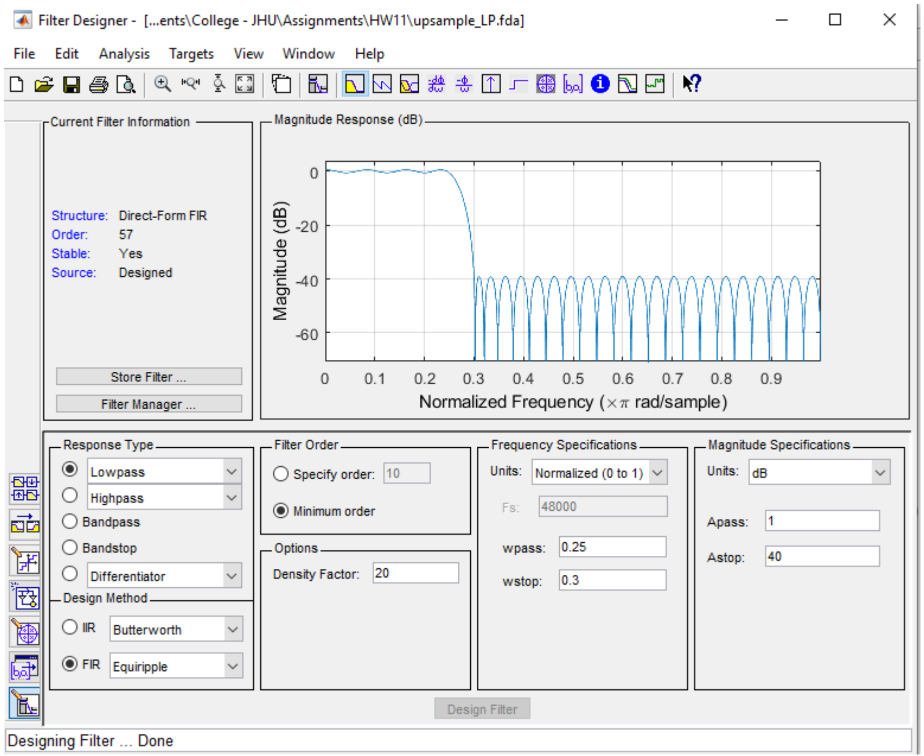


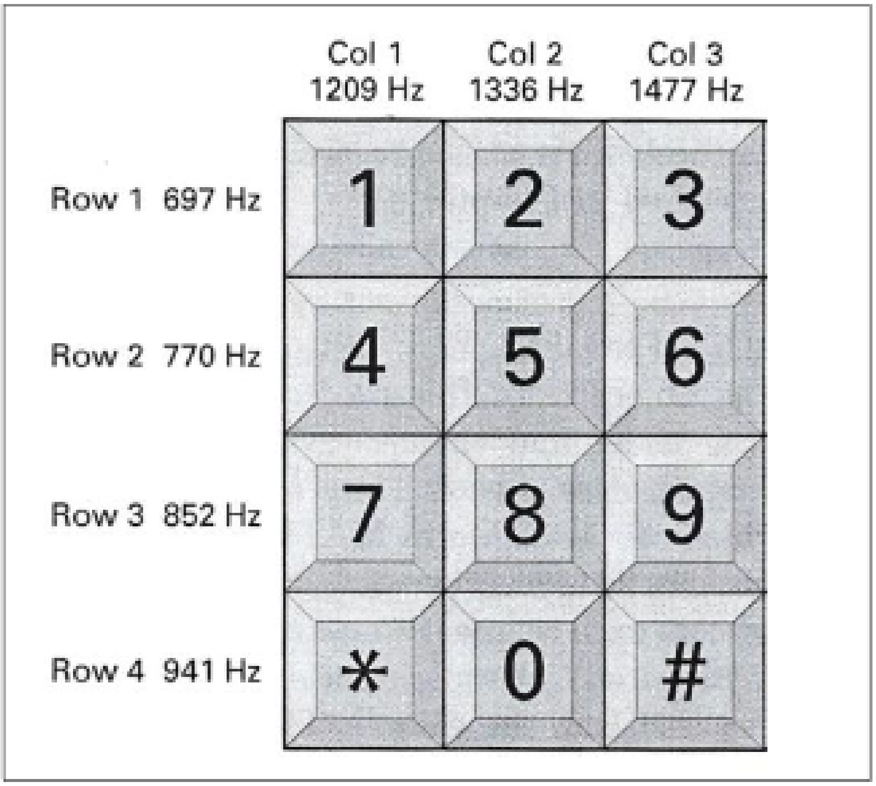
Figure 3: Lowpass filter for removing spectral copies.

the tones produced by the keypad and dtmf function. As the designer you select the type of filter and bandwidth for each filter.

Use the filter visualization tool fvtool shown below to analyze and document filter frequency response of each filter. Also note and record the group delay near the filter's center frequency. This can be helpful in setting the timing in sampling frequency bursts to determine their presence in the output of the respective filters.

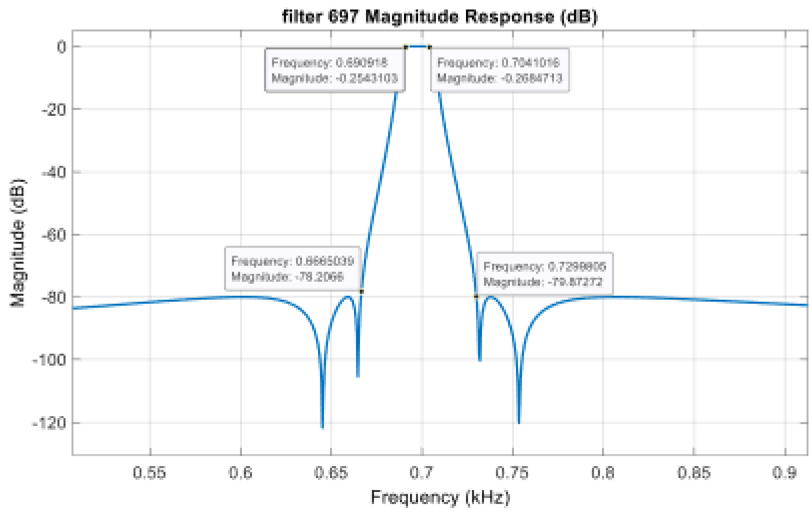
The output from these filters is squared to get a DC component proportional to the power together with unwanted higher frequency signals. We then need to filter out the higher frequency components.This is accomplished by passing the outputs from the filter through a followon LP filter like the one shown below in the filterDesigner window.

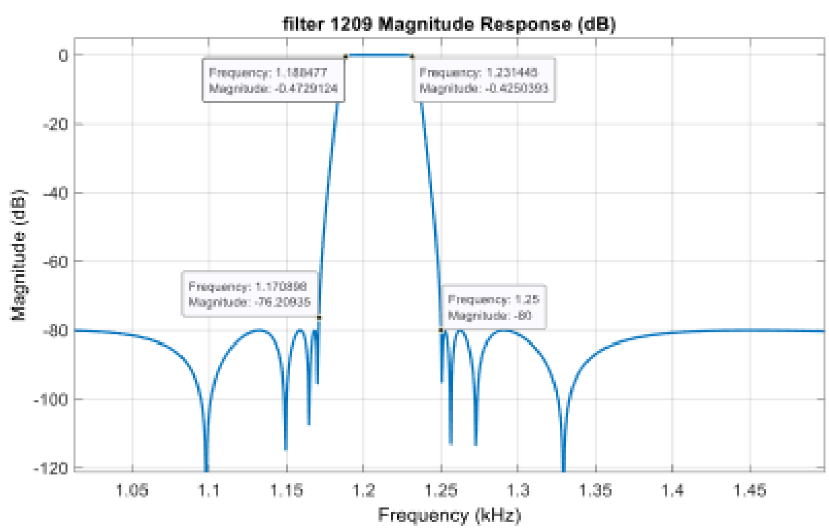
The presence of the power associated with each filter designated using the letter y in the variable name is then quantized using a commands like (y697out¿.3)or (y1209out¿.3)which creates a Boolean output of 0 or 1 depending on whether the condition is false or true. The Boolean value can be changed to a double preci-

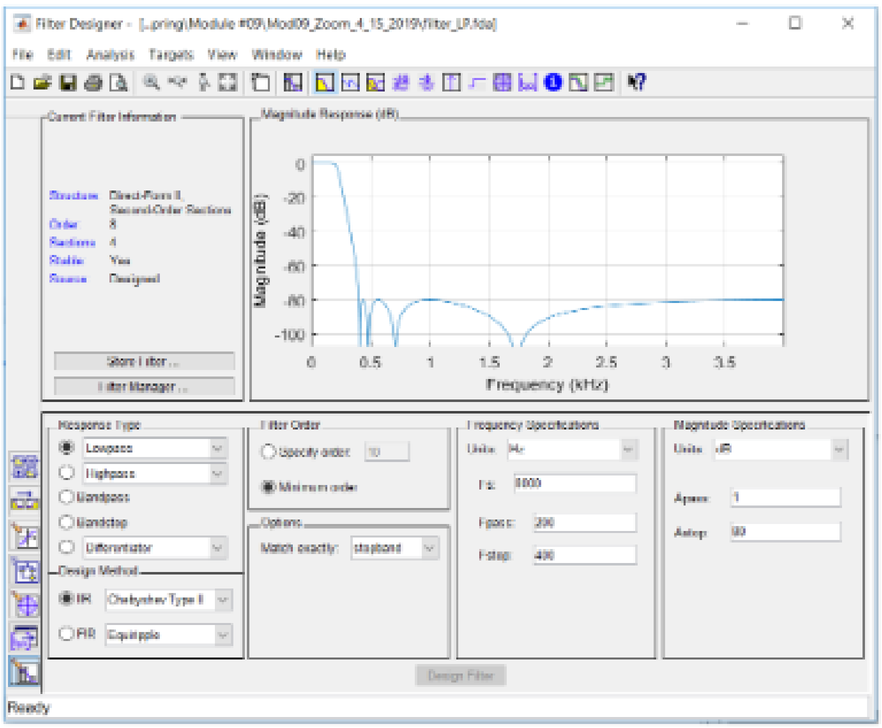


sion value using the command double( ) as you will see. These results are stored in an array and then are downsampled to detect the tone centers state.This is where the group velocity and/or figure 1 can help determine the downs ample offset necessary to correctly determine whether a dtfm signals is present for each filter.These results are used to determine the row and column associated with a given tone and this determines the data value. It is then accumulated as a vector giving the output values. The attached example script file illustrates this process to assist you in carryout the required assignment specified below.

```
function [x,Fs] = dtmf(phone_num)
% dtmf(phone_num)
%
% phone_num is a vector containing the number to be called.
% "*" and "#" are represented as "-1" and "-2", respectively.

Fs = 8000;
T = 0.25;
tau = 0.05;

N = length(phone_num);
tt = (0:1/Fs:T)';
K = length(tt);
M = round(Fs*(T+tau));

x = zeros(ceil(N*Fs*(T+tau)),1);

for n = 1:N
  index = ((n-1)*M + 1:(n-1)*M + K);

  switch phone_num(n)
  case 1; x(index) = x(index) + sin(2*pi*697*tt) + sin(2*pi*1209*tt);
  case 2; x(index) = x(index) + sin(2*pi*697*tt) + sin(2*pi*1336*tt);
  case 3; x(index) = x(index) + sin(2*pi*697*tt) + sin(2*pi*1477*tt);
   case 4;x(index) = x(index) + sin(2*pi*770*tt) + sin(2*pi*1209*tt);
   case 5;x(index) = x(index) + sin(2*pi*770*tt) + sin(2*pi*1336*tt);
   case 6;x(index) = x(index) + sin(2*pi*770*tt) + sin(2*pi*1477*tt);
   case 7;x(index) = x(index) + sin(2*pi*852*tt) + sin(2*pi*1209*tt);
   case 8;x(index) = x(index) + sin(2*pi*852*tt) + sin(2*pi*1336*tt);
   case 9;x(index) = x(index) + sin(2*pi*852*tt) + sin(2*pi*1477*tt);
  case -1;x(index) = x(index) + sin(2*pi*941*tt) + sin(2*pi*1209*tt);
   case 0;x(index) = x(index) + sin(2*pi*941*tt) + sin(2*pi*1336*tt);
  case -2;x(index) = x(index) + sin(2*pi*941*tt) + sin(2*pi*1477*tt);
  otherwise
    disp('unknown number');
  end
```


filter 697 Magnitude Response (dB)
Frequency: 0.690918
Magnitude: -0.2543103
Frequency: 0.7041016
Magnitude: -0.2684713
Frequency: 0.6665039
Magnitude: -78.2066
Frequency: 0.7299805
Magnitude: -79.87272
Magnitude (dB)
Frequency (kHz)
0
-20
-40
-60
-80
-100
-120
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9


filter 1209 Magnitude Response (dB)
Frequency: 1.188477
Magnitude: -0.4729124
Frequency: 1.231445
Magnitude: -0.4250393
Frequency: 1.170898
Magnitude: -76.20935
Frequency: 1.25
Magnitude: -80
Magnitude (dB)
Frequency (kHz)
0
-20
-40
-60
-80
-100
-120
1.05
1.1
1.15
1.2
1.25
1.3
1.35
1.4
1.45


Filter Designer - [..pring\Module #09\Mod09_Zoom_4_15_2019\filter_LP.fda]
File Edit Analysis Targets View Window Help
Current Filter Information
Structure Direct-Form II, Second-Order Sections
Order 8
Sections 4
Stable Yes
Source Designed
Store Filter ...
Filter Manager ...
Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR Chebyshev Type II
FIR Equiripple
Filter Order
Specify order: 10
Minimum order
Options
Match exactly: stopband
Frequency Specifications
Units: Hz
Fs: 8000
Fpass: 200
Fstop: 400
Magnitude Specifications
Units: dB
Apass: 1
Astop: 80
Design Filter
Ready

Part A

Students are select filter types and properties and then design filters to detect 7 dtfm filters signal and a low pass filter for output power detection. Display and document properties using MATLAB filter visualization tool(fvtool) which can be found in the filterDesigner "view" menu.

For my filter, I selected an Elliptic filter design with cutoff frequencies $F_c = F_{tone} \pm 20Hz$ and a stopband frequency $F_{stop} = F_{tone} \pm .70Hz$. I chose these frequencies since the smallest ΔHz between the dial tones was 73Hz. I specifically chose an IIR Elliptic filter since it has a better transition band out of the other options (Butterworth, Chebychev I/II, etc.) that was available in filterDesigner. FIR filters are nice for anything involving timing, but for my filter specs I would have had a 300+ filter order vs. the filter order of 8. Elliptic filters do have a disadvantage of having a very non-linear phase delay, so we will have to see how this handles. The objective is to simply detect the tones, and if we know the group delay of the filter we can at least compensate if needed.

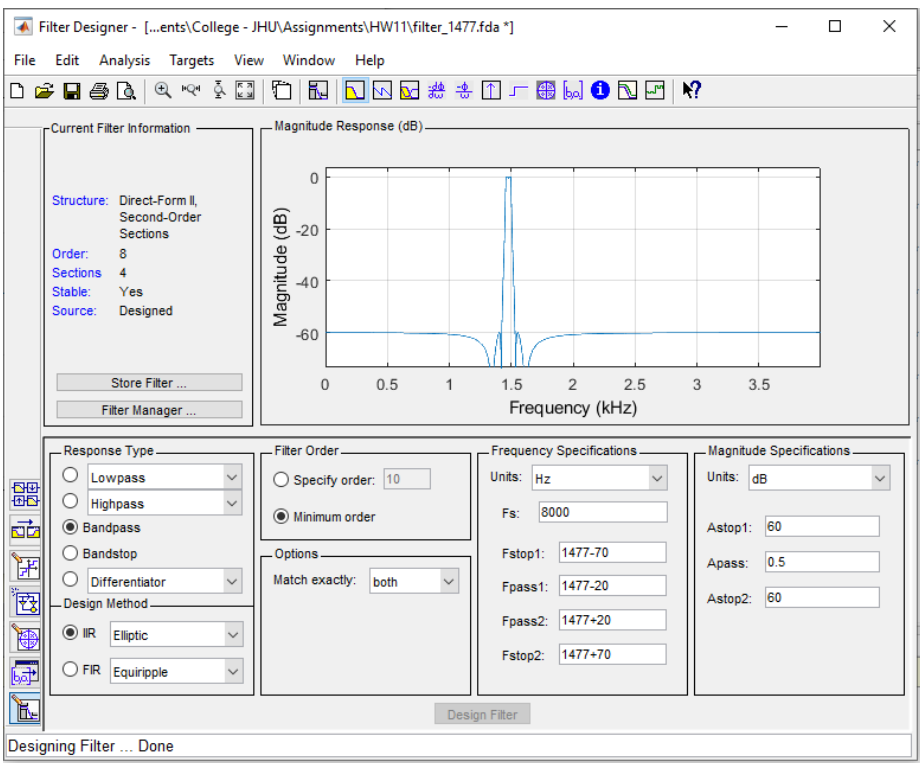


Figure 4: Filter Designer window with specifications for this task

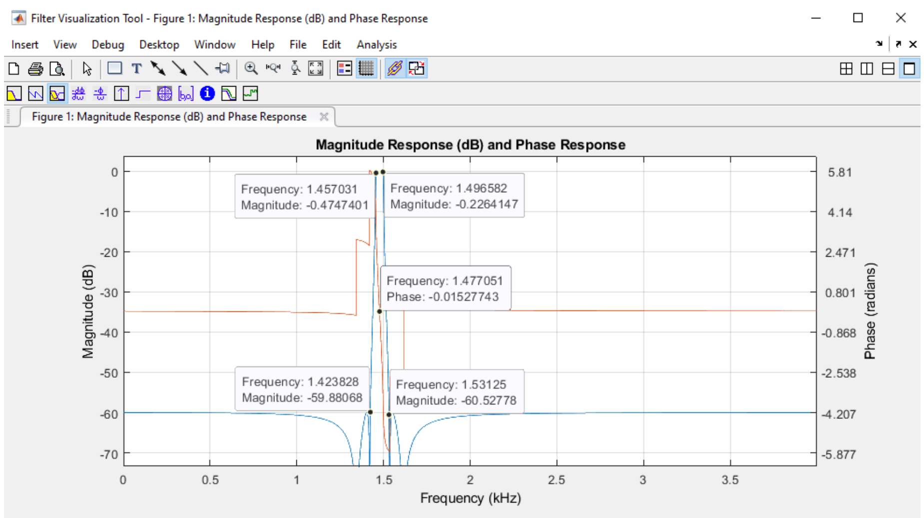


Figure 5: Filter Visualization Tool

Part B

Export each filter design as a filter object to MATLAB workspace

See MATLAB code below.

Part C

*Save each filter object to computer folder as a *.mat file*

Can be done by going to file → export. Export to a .mat file and the coefficients as "objects". Name the .mat file accordingly.

Part D

Determine down sampling offset to determine presence of filter output power

Since the phase offset is very close to zero for the center frequencies of each dial tone, I set the offset to 1250. Using the help functionality I found that the phase has to be set as an integer from range $[0, N-1]$. Since $N = 2500$, I selected half of N to get a phase offset of zero.

Part E

Run the scriptfileto detect and display the data sequence 1,2,3,4,5,6,7,8,9,0 . Print out the script file along with the MATLAB output displays.

Below in Figures 6,7 and 8 are the results.

```
Transmitted Data =  1  2  3  4  5  6  7  8  9  0
```

Figure 6: Initial Phone Number

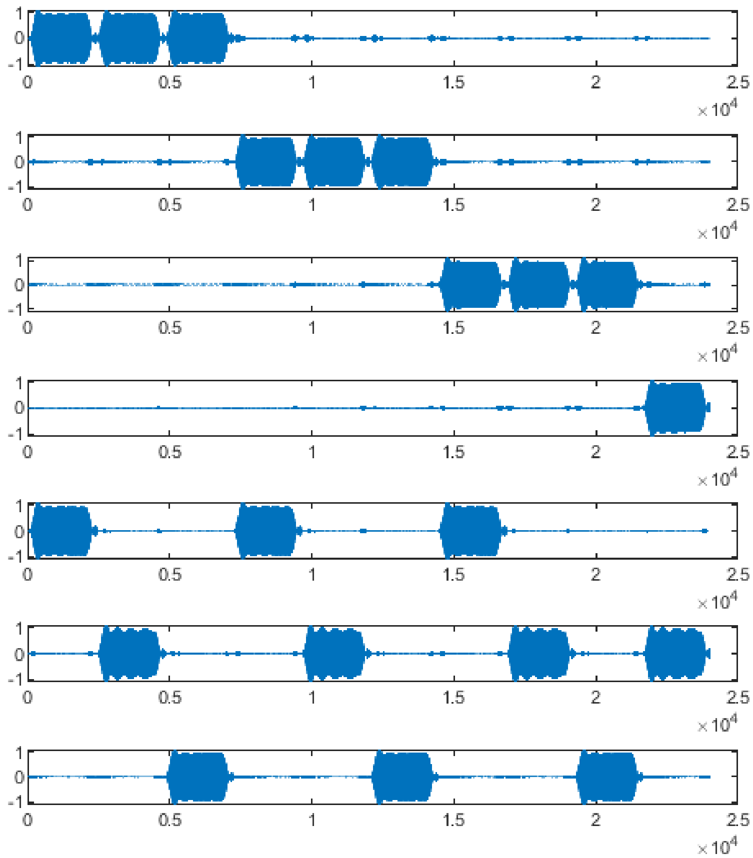


Figure 7: Pulse Tones Generated

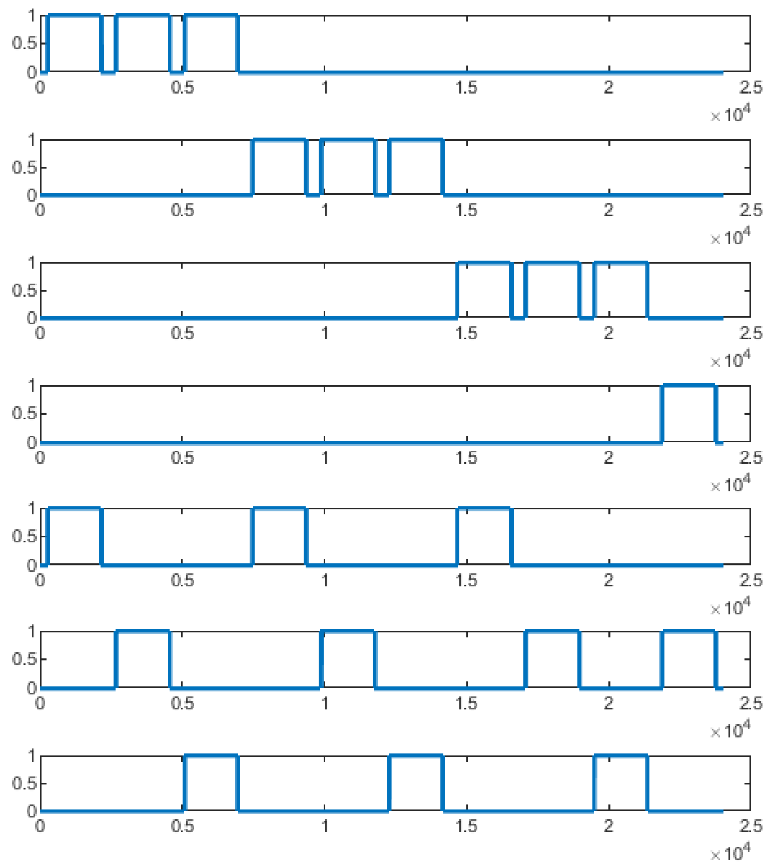


Figure 8: Detection of Dial Tones

Matlab Code

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem 11.1
%%%%%%%%%%%%%%%%%%%%%%%%%%
% create signal x(n), representing 8 sinusoidal tones
N = 2048;
n=0:N-1;          % signal index
nseg=1000:1060;
Hz=1; KHz=1e3;
Fs = 4000*Hz;    % Samples/sec
Ts = 1/Fs;

% Analog Signal Generation
F = [100:50:450].';
A1 = [1:4]/4; A=[A1,fliplr(A1)];
xn=A*sin(2*pi*F*n*Ts);

% Creation of Analog Time Index
ms = 1e3;
duration = N/Fs * ms;
t1 = linspace(0,duration,N);


% -- Part A --
% Plot analog representation of our created signal x(n)
figure(1)
subplot(5,1,1);
plot(t1,xn);
xlabel('Time (ms)');
xlim([0,duration]);
ylabel('Amplitude');
title('Band-Limited Analog Signal x(t)')

% -- Part B --
% Show sampled representation of a section of x(n)
t2 = [1000:1160]; % nseg index for time slot window

subplot(5,1,2);
stem(t2,xn(1000:1160));
xlabel('Sample n');
ylabel('Amplitude');
title('Sampled Signal x(n)')

% -- Part C --
```

```
% Plot the spectrum of signal x(n)
n = linspace(-N/2,N/2,N);
w = linspace(-2*pi,2*pi,N);
X = dtft(xn,n,w)/N;

subplot(5,1,3);
plot(w/pi,20*log10(abs(X)))
title('Spectrum Magnitude |X(\omega)|')
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')

% --- Part D ---
% Downsample signal x(n)
yn = downsample(xn,4);

% Creation of Downsampled Analog Time Index
ms = 1e3;
DFs = Fs/4; % downsampled FS
t3 = linspace(0,duration,N/4);

subplot(5,1,4);
% NOTE: the index is converted to ms
stem(t3(125:140),yn(125:140));
xlabel('Sample n');
ylabel('Amplitude');
title('Signal x(n) Downsampled by 4')

% --- Part E ---
% Compute and display the DTFT of downsampled x(n)
Nd = N/4;
nd = linspace(-Nd/2,Nd/2,Nd);
wd = linspace(-2*pi,2*pi,Nd);
Xd = dtft(yn,nd,wd)/Nd;

subplot(5,1,5);
plot(wd/pi,20*log10(abs(Xd)))
title('Spectrum Magnitude of Downsampled |X(\omega)|')
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')


%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem 11.2
%%%%%%%%%%%%%%%%%%%%%%%%%

% --- Part A ---
```

```
% Reproduce the original signal x(t), and compare to
    interpolated x(n)
x_interp = interp(yn,4);

samples = xn';
tr = -10:0.05:N+10;
xr = zeros(N,length(tr));
% -- Part A --
% Reconstructing x(t) from downsampled x(n)


for n = 0:N-1
    xr(n+1,:) = samples(n+1)*sinc((tr-n));
end

figure(2)
subplot(3,1,1)
plot(t1,x_interp,'r','linewidth',2)
xlabel('Time (ms)');
xlim([0,duration]);
ylabel('Amplitude');
title('Interpolation of x(n) by 4')

twhat = linspace(0,duration,length(tr));

subplot(3,1,2)
plot(tr,sum(xr),'r','linewidth',2)
xlabel('Time (ms)');
xlim([0,duration]);
ylabel('Amplitude');
title('Reconstruction of Analog Signal x(t)')

% -- Part B --
% Reconstruct x(t) from y(n), our downsampled signal of x
    (n)
samples = yn';
% tr = 0:0.05:duration;
tr = -10:0.05:N/4+10;
yr = zeros(N/4,length(tr));
% -- Part A --
% Reconstructing x(t) from downsampled x(n)
figure(2)

for n = 0:N/4-1
    yr(n+1,:) = samples(n+1)*sinc((tr-n));
end
```

```matlab

twhat = linspace(0,duration,length(tr));

subplot(3,1,3)
plot(tr,sum(yr),'r','linewidth',2)
xlabel('Time (ms)');
xlim([0,duration]);
ylabel('Amplitude');
title('Reconstruction of y(n)')

%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem 11.3
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Below is given code that shows our original signal, a segment, and the
% computed dtft for comparison against an upsampled signal
close all
clear
clc
N=2048; n=0:N-1; nseg=1000:1060;
Hz=1;KHz=1e3; Fs=4000*Hz;
Ts=1/Fs;

F=[100:50:450].'; A1=[1:4]/4; A=[A1,fliplr(A1)];
xn=A*sin(2*pi*F*n*Ts);
figure(1)
Pos=get(1,'Position'); set(1,'Position',Pos+[0 -300 0 300])
subplot(3,2,1)
plot(n*Ts*1000,xn,'LineWidth',.5);grid on
set(gca,'FontSize',14,'Xlim',[0 512])
title('Signal (xn) over its full time range');
xlabel('Time (msec)');
ylabel('Volts')

subplot(3,2,2)
stem(nseg*Ts*1000,xn(nseg),'LineStyle','none',...
    'MarkerSize',2,'LineWidth',2),grid on
set(gca,'FontSize',14)
title(['Original Signal at sample rate = ',...
    num2str(Fs/1000),' KHz']);
xlabel('Expanded Sample Time Scale (msec)');
ylabel('Volts')

subplot(3,2,3)
```

```
wx=[-2000:2000]*pi/1000;
X=dtft(xn,n,wx)/length(n); magX=abs(X);
plot(wx/pi,magX,'r','LineWidth',1);grid on
set(gca,'FontSize',14,'Ylim',[0 .6])
title('DTFT Original Signal');
xlabel('Radian Frequency (\omega_x/\pi)');
ylabel('|dtft|')


% -- Part C --
% Upsample x(n) by 4 samples and plot
yn = upsample(xn,4);
nseg=4000:4240;

subplot(3,2,4)
stem(4*nseg*Ts*1000,yn(nseg),'LineStyle','none',...
    'MarkerSize',2,'LineWidth',2),grid on
set(gca,'FontSize',14)
title(['Upsampled Signal at sample rate = ',...
    num2str((4*Fs)/1000),' KHz']);
xlabel('Expanded Sample Time Scale (msec)');
ylabel('Volts')

% -- Part D --
% Compute the DTFT of y(n)
subplot(3,2,5)
n=0:4*N-1;
wy=[-2000:2000]*pi/1000;
Y=dtft(yn,n,wy)/length(n); magY=abs(Y);
plot(wy/pi,magY,'r','LineWidth',1);grid on
set(gca,'FontSize',14,'Ylim',[0 .6])
title('DTFT Upsampled Signal');
xlabel('Radian Frequency (\omega_x/\pi)');
ylabel('|dtft|')

% -- Part E --
% Import the designed lowpass filter, filter y(n), and then plot.
load upsample.mat

yn_lpf = filter(Hd,yn);

Y=dtft(yn_lpf,n,wy)/length(n); magY=abs(Y);
subplot(3,2,6)
plot(wy/pi,magY,'r','LineWidth',1);grid on
set(gca,'FontSize',14,'Ylim',[0 .6])
```

```
title('DTFT Upsampled Signal');
xlabel('Radian Frequency (\omega_x/\pi)');
ylabel('|dtft|')
%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem 11.4
%%%%%%%%%%%%%%%%%%%%%%%%%
% Most of this code was given. See hw11 documentation.
close all
clear
clc

phone_num=[1 2 3 4 5 6 7 8 9 0];
[x,Fs] = dtmf(phone_num);
N=2400;
n=[1:2*N-1];

%filterDesigner -->export filter object-->save to file-->
    load
load filter_697.mat
load filter_770.mat
load filter_852.mat
load filter_941.mat
load filter_1209.mat
load filter_1336.mat
load filter_1477.mat

y697=filter(filter_697,x);
y770=filter(filter_770,x);
y852=filter(filter_852,x);
y941=filter(filter_941,x);
y1209=filter(filter_1209,x);
y1336=filter(filter_1336,x);
y1477=filter(filter_1477,x);
load filter_LP.mat

figure(1)
Pos=get(1,'Position');
set(1,'Position',Pos+[0 -200 0 200])

subplot(7,1,1),plot(y697)
subplot(7,1,2),plot(y770)
subplot(7,1,3),plot(y852)
subplot(7,1,4),plot(y941)
subplot(7,1,5),plot(y1209)
subplot(7,1,6),plot(y1336)
subplot(7,1,7),plot(y1477)
```

```

y697out=filter(filter_LP,y697.^2);    y(1,:)=double(y697out>.3);
y770out=filter(filter_LP,y770.^2);    y(2,:)=double(y770out>.3);
y852out=filter(filter_LP,y852.^2);    y(3,:)=double(y852out>.3);
y941out=filter(filter_LP,y941.^2);    y(4,:)=double(y941out>.3);
y1209out=filter(filter_LP,y1209.^2); y(5,:)=double(y1209out>.3);
y1336out=filter(filter_LP,y1336.^2); y(6,:)=double(y1336out>.3);
y1477out=filter(filter_LP,y1477.^2); y(7,:)=double(y1477out>.3);

figure(2)
Pos=get(2,'Position');
set(2,'Position',Pos+[0   -200    0     200])

for p=1:7
    subplot(7,1,p), plot(y(p,:),'LineWidth',2)
end

for q=1:7
    %can use group delay or figure 1 as an offset guide setting XXX
    d(q,:)=downsample(y(q,:),2500,1250); % 1250 = offset of zero
end

% disp(num2str(d))
numarray=[1 2 3;4 5 6; 7 8 9; -1 0 -2];
num=zeros(1,length(x)/2400);
for r=1:10
    index=find(d(:,r)==1);
    row= index(1); col=index(2)-4;
    num(1,r)=numarray(row,col);
end
disp(' ')
disp(['  Transmitted Data =  ',num2str(num)])
```