ELEC 466 : DIGITAL SIGNAL PROCESSING
1995/96 WINTER SESSION TERM 2

Solutions to Assignment 9
Finite Precision and Multirate DSP

Question 1

The 8-th order Elliptic bandpass filter was designed in Question 3 in the previous assignment using the commands:

```
>> [b a]=ellip(4,0.5,60,[0.5/25 22/25]);
```

We can scale the coefficients so their maximum absolute value is 1 using the commands:

```
>> b=b/max(abs(b));
>> a=a/max(abs(a));
```

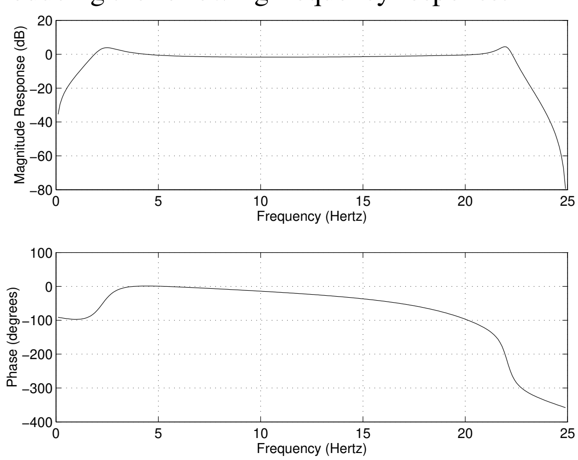
To quantize the filter coefficients to 6 bits assuming one of the 6 bits is required for a sign bit:

```
>> L=2^(6-1);
>> b6=round(b*L)/L;
>> a6=round(a*L)/L;
```

The amplitude response can be evaluate using:

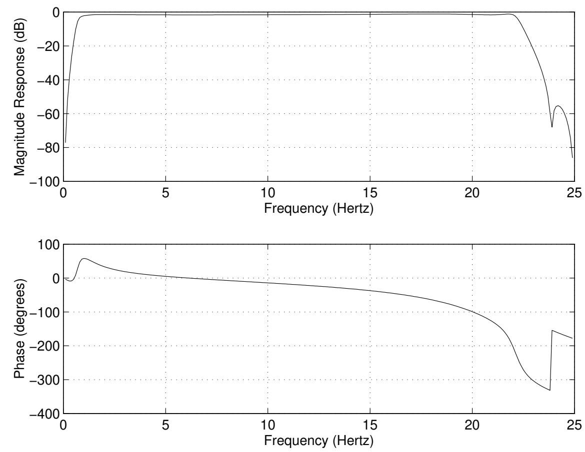
```
>> freqz(b6,a6,[0.1:0.1:24.9],50);
>> print -deps sol9.q1a.eps
```

producing the following frequency response:



For 12 bits the commands and resulting frequency response are:

```
>> L=2^(12-1);
>> b12=round(b*L)/L;
>> a12=round(a*L)/L;
>> freqz(b12,a12,[0.1:0.1:24.9],50);
>> print -deps sol9.q1b.eps
```



Question 2

The poles and zeros of the above (unquantized) filter can be found using Matlab's `roots()` function:

```
>> [b a]=ellip(4,0.5,60,[0.5/25 22/25]);
>> z=roots(b);
>> p=roots(a);
```

A cascade realization of this filter will require using four 2nd-order IIR sections. To find the coefficients for these filters we expand pairs of factors using Matlab's `poly()` function (note that we pair complex conjugate roots to ensure real coefficients):

```
>> b1=poly(z(1:2));
>> b2=poly(z(3:4));
>> b3=poly(z(5:6));
>> b4=poly(z(7:8));
>>
>> a1=poly(p(1:2));
>> a2=poly(p(3:4));
```

```
>> a3=poly(p(5:6));
>> a4=poly(p(7:8));
```

We then quantize the coefficients of each second order section to 6 bits, scaling so the largest coefficient is 1 as before using the following commands:

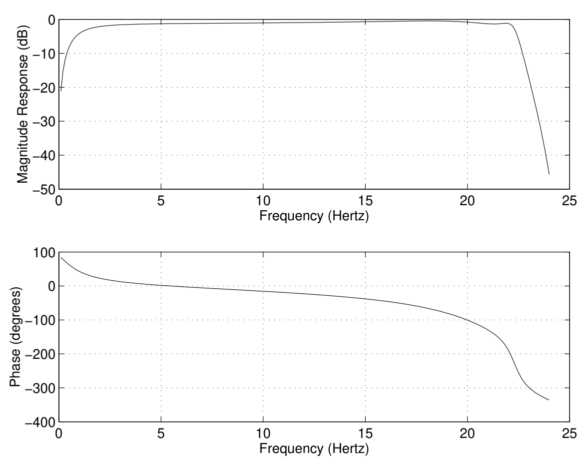
```
>> L=2^(6-1);
>> b1=round( b1./max(abs(b1)) * L )/L;
>> b2=round( b2./max(abs(b2)) * L )/L;
>> b3=round( b3./max(abs(b3)) * L )/L;
>> b4=round( b4./max(abs(b4)) * L )/L;
>>
>> a1=round( a1./max(abs(a1)) * L )/L;
>> a2=round( a2./max(abs(a2)) * L )/L;
>> a3=round( a3./max(abs(a3)) * L )/L;
>> a4=round( a4./max(abs(a4)) * L )/L;
```

Finally, we find the coefficients of the cascade realization of filters with quantized coefficients by computing coefficients of the cascade realization using the Matlab `series()` function:

```
>> [bb aa]=series(b1,a1,b2,a2);
>> [bb aa]=series(bb,aa,b3,a3);
>> [bb aa]=series(bb,aa,b4,a4);
```

and plot the frequency response of the overall filter:

```
>> freqz(bb,aa,[0.1:0.1:24],50);
>> print -deps sol9.q2.eps
```



Question 3

A sine wave with a frequency of $F_s/3$ can be generated using:

```
>> x=sin(2*pi*1/3*[0:15]);
```

where we have set $F_s = 1$.

To increase the sampling rate by a factor of 6 we create a vector of 6×16 zeros, and insert one sample very 6 values:

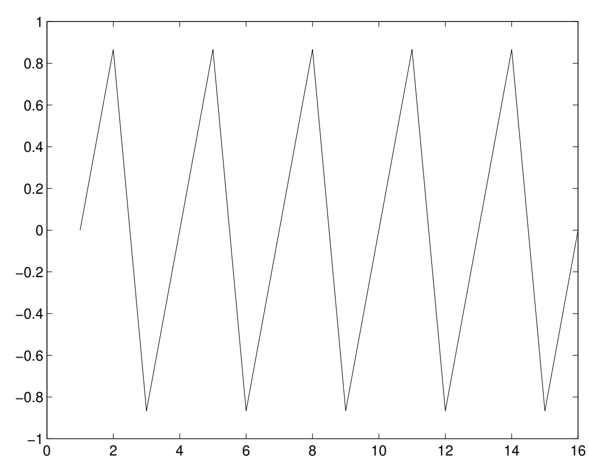
```
>> y=zeros(1,96);
>> y(1:6:91)=x(1:16);
```

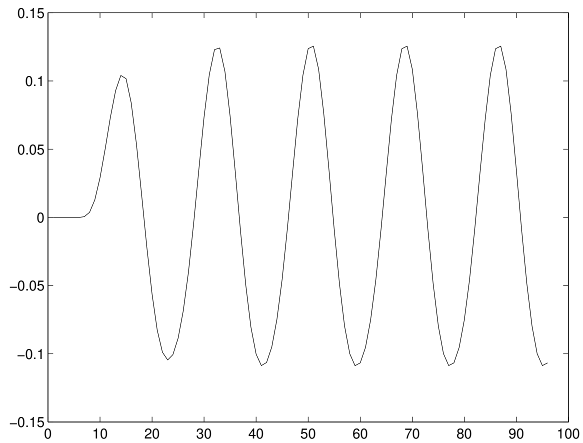
This increases the sampling rate to 6. A 4-th order Butterworth interpolation filter with a -3 dB frequency of $F_s/3 = 1/3$ can be designed using (note that the frequency argument must be normalized by dividing by $F_s/2 = 3$):

```
>> [b a]=butter(4,(1/3)/3);
```

We can then filter and plot the original and the interpolated waveforms using:

```
>> y=filter(b,a,y);
>> plot(x);
>> print -deps sol9.q3a.eps
>> plot(y);
>> print -deps sol9.q3b.eps
```





The change in amplitude is due to the attenuation of the interpolation filter (a factor of $1/\sqrt{2}$) and the fact that interpolator inserts 5 zeros between each sample thus reducing the average value by a factor of 1/6 giving an amplitude of about 0.12.

Question 4

(a)

There are three possible implementations for a rate-6 decimator:

1. a single rate-6 decimator
2. a rate-3 decimator followed by a rate-2
3. a rate-2 decimator followed by a rate-3

To find the number of operations required for the filters in each of the three possible implementations we first need to find the order of each filter using `remezord()`. To do this we need to know the sampling rate that each filter operates at, the passband and stopband ripple, and the passband and stopband edge frequencies.

In each case the passband will include frequencies from 0 to 3 kHz and the edge of the stopband will be at the frequency where aliasing would happen (the "aliasing frequency"). Since the subsampling operation causes the spectrum to be replicated about multiples of the decimator output sampling rate, this aliasing frequency will be the output sampling rate minus the stopband edge frequency (4 kHz in this case).

The overall ripple in the passband must be ± 1 dB. For the two-stage implementations each stage should have a ripple of ± 0.5 dB to ensure the worst-case ripple is ± 1 dB. In each case the stopband ripple must be the same (-40 dB) since new aliasing components are introduced at each stage.

In linear units $\pm 1\text{dB} = 10^{1/20} - 10^{-1/20} = \pm 0.23$, $\pm 0.5\text{dB} = 10^{0.5/20} - 10^{-0.5/20} = \pm 0.115$, and $-40\text{dB} = 10^{-40/20} = 0.01$. In the computations below we'll use peak passband ripple values (half of the peak-to-peak values).

1. For a single stage decimator the sampling frequency at the output of the decimator is 8 kHz. The aliasing frequency is $8 - 4 = 4$ kHz. The filter order and number of operations for this implementation is:

```
>> [N Fo Mo W]=...
 remezord([3 4],[1 0],[0.115 0.01],48);
>> N
N =
    58
>> N*48
ans =
        2784
```

2. For a two-stage decimator with the first stage dividing by 3, the output sampling frequency of the first stage will be 16 kHz. The aliasing frequency is 16-4=12 kHz. The sampling frequency at the output of the second stage will be 8 kHz so the aliasing frequency is 8-4=4 kHz. The filter orders and number of operations for this implementation is:

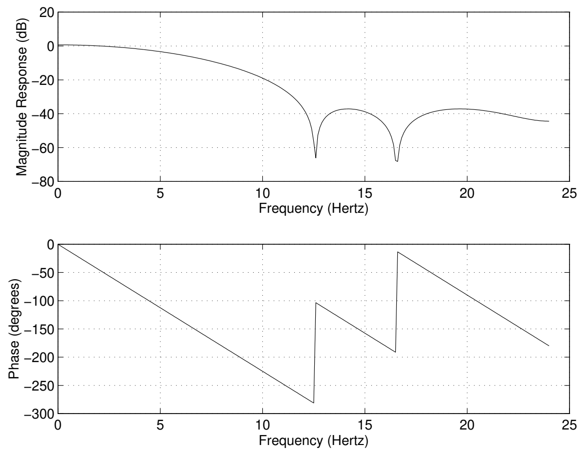
```
>> [N1 Fo Mo W]=...
 remezord([3 12],[1 0],[0.058 0.01],48);
>> N1
N1 =
     6
>> [N2 Fo Mo W]=...
 remezord([3 4],[1 0],[0.058 0.01],16);
>> N2
N2 =
    22
>> N1*48+N2*16
ans =
   640
```

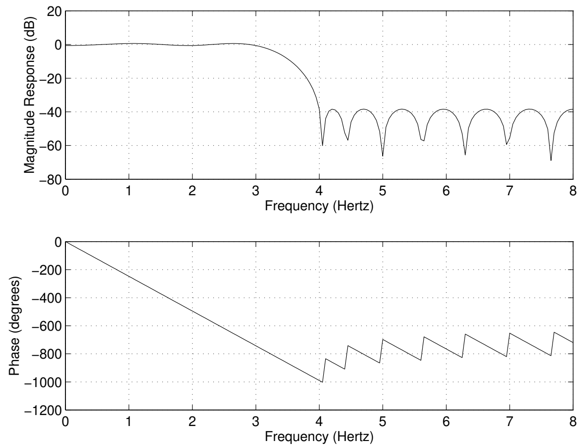
3. For a two-stage decimator with the first stage dividing by 2 the output sampling frequency of the first stage will be 24 kHz and the aliasing

frequency is 24-4=20 kHz. The sampling frequency at the output of the second stage will be 8 kHz so the aliasing frequency is 8-4=4 kHz. The filter orders and number of operations for this implementation is:

```
>> [N1 Fo Mo W]=...
 remezord([3 20],[1 0],[0.058 0.01],48);
>> N1
N1 =
      0
>> [N2 Fo Mo W]=...
 remezord([3 4],[1 0],[0.058 0.01],24);
>> N2
N2 =
    34
>> N1*48+N2*24
ans =
   816
```

Recall that `remezord()` only provides an estimate of the required order. Obviously the estimate for `N1` in this case is quite far off.





(b)

The most efficient design reduces the sampling rate by 3 in the first stage. We can design the filters and plot the frequency response using the following Matlab commands:

```
>> [N Fo Mo W]=...
 remezord([3 12],[1 0],[0.058 0.01],48);
>> B=remez(N,Fo,Mo,W);
>> freqz(B,[1],[0:0.1:24],48);
>> print -deps sol9.q4b1.eps
>> [N Fo Mo W]=...
 remezord([3 4],[1 0],[0.058 0.01],16);
>> B=remez(N,Fo,Mo,W);
>> freqz(B,[1],[0:0.05:8],16);
>> print -deps sol9.q4b2.eps
```

The filter responses are as follows: