ELECTRONOTES
1 Pheasant Lane
Ithaca, NY 14850
(607)-273-8030

APPLICATION NOTE NO. 337

March 1996

A GENERAL REVIEW OF FREQUENCY SAMPLING DESIGN

1. INTRODUCTION

The z-transform of a digital filter's impulse response, h(n), is the filters transfer function H(z) where:

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)\, z^{-n} \qquad (1)$$

For an FIR filter, chosen so that h(n) is zero except for n=0,1,...N-1, and for H(z) evaluated on the unit circle in the z-plane ($z = e^{j\omega}$) we have:

$$H(e^{j\omega}) = \sum_{n=0}^{N-1} h(n)\, e^{-jn\omega} \qquad (2)$$

which is the Discrete-Time Fourier Transform (DTFT) of the impulse response, commonly referred to as the frequency response. Note that $H(e^{j\omega})$ is a continuous function of frequency ω, periodic in ω with period 2π, even though h(n) is discrete in time. Thus $H(e^{j\omega})$ and h(n) in equation (2) constitute a "Fourier series" expansion, although the roles of time and frequency are interchanged relative to their usual presentation (where a periodic function of time is represented in terms of discrete frequency components). Here the discrete time function has a periodic description in frequency.

Although $H(e^{j\omega})$ is continuous, it can be evaluated at discrete points. If we evaluate it at N discrete, equally spaced frequency points, indexed by k=0,1,2,... N-1, we have frequencies:

$$\omega_k = (2\pi/N)k \qquad (3)$$

and we can write:

$$H(k) \equiv H(e^{j\omega k}) = H(e^{j(2\pi/N)k}) = \sum_{n=0}^{N-1} h(n) e^{-j(2\pi/N)nk} \qquad (4)$$

which is the DFT of h(n).

This suggests the simplest view of frequency sampling where the impulse response is obtained by taking

$$h(n) = DFT^{-1}[D(k)] \tag{5}$$

where the D(k) are samples of some desired frequency response. Note that because a DFT is the same as samples of the DTFT, the frequency response of the filter obtained in this way will match $D(e^{j\omega})$ exactly at the points ω_k, but not in general elsewhere. The idea is that if the samples are taken closely enough together, the errors between the samples may not be too large.

2. IGNORING THE PHASE

Equation (5) is not a general case of frequency sampling, and yet it is already deceptively simple. In particular, we often think of a desired response in terms of the desired magnitude of the response, but here we must also get the phase correct, and this can be a problem. It is interesting to see how bad thing can get.

Suppose we are negligent about phase. We decide to take frequency samples purely on the basis of magnitude. That is, we want, for example, a low-pass with a cutoff at 1/4 the sampling frequency. Suppose further that we decide on an N=31 length filter. For sampling frequency f_s, the magnitude of the samples could be chosen at frequencies 0, $f_s/31$, $2f_s/31$, $3f_s/31$, and so on up to $30f_s/31$. Accordingly samples for frequencies $kf_s/31 < f_s/4$ (i.e., for k=0 1,2,...7) will be 1. It might seem that samples for k=8 and above should be zero, but we know that the frequency response has a negative side and is periodic in f_s. Thus a samples at f_s (which is not included here) has magnitude of 1, as do seven more samples (which are included) below f_s. Thus we also have ones at $30f_s/31$, $29f_s/31$, $28f_s/31$, $27f_s/31$ $26f_s/31$, $25f_s/31$, and $24f_s/31$. All other samples have zero magnitude. Thus we might try (and will be unsuccessful with) a D(k) as:

$$\begin{aligned} D(k) &= 1 \quad k=0,1,2,3,4,5,6,7 \\ &= 0 \quad k=8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23 \\ &= 1 \quad k=24,25,26,27,28,29,30 \end{aligned} \tag{6}$$

If we take h(n) as the inverse DFT of this D(k), we get the impulse response shown in Fig. 1a and the corresponding magnitude of the frequency response shown in Fig. 1b. Clearly, this is almost certainly not what we had in mind as a low-pass filter. However, note that the magnitude response does in fact go through the specified samples.

While it is clear from the magnitude response that the impulse response we have obtained is unsatisfactory, it may also be clear from looking at the impulse response directly that it is not low-pass. For a low-pass, we would expect the impulse response to have a sinc-like profile (larger impulse response values

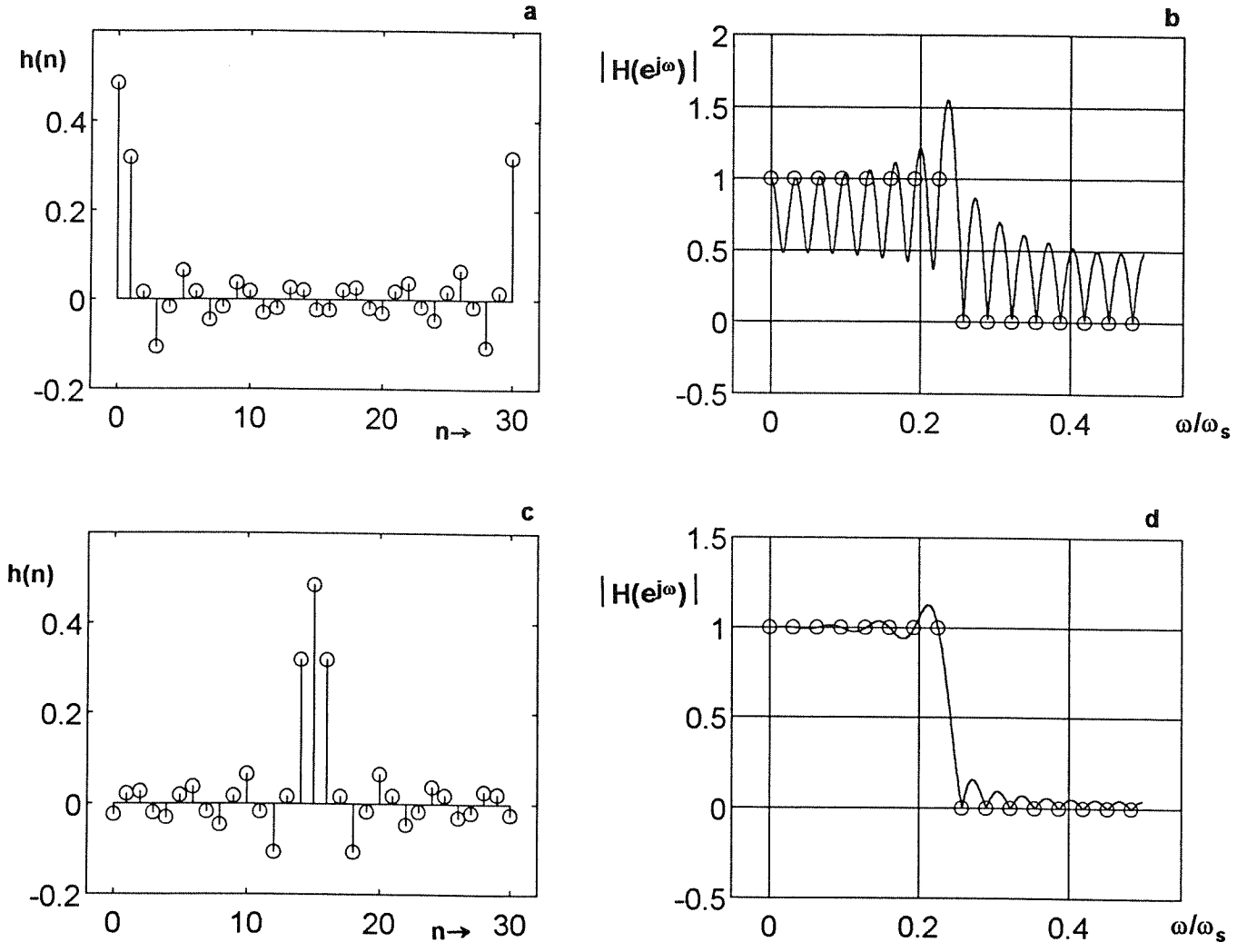


Fig. 1 Zero phase filter has impulse response as seen in Fig. 1a and magnitude response as shown in Fig. 1b. Rotating the impulse response to the sinc-like response of Fig. 1c results in an acceptable low-pass magnitude as seen in Fig. 1d. This is only an ad hoc fix for this particular case, however.

concentrated in the middle). Here we see that the impulse response peaks at the ends. [In fact, it is this peaking at the ends that accounts for the ripples.] It is probably evident that if we were to take n=0 in Fig. 1a as the center, wrapping the response around periodically, that we would get something better resembling a sinc-like impulse response. This is indeed the case. In fact, if we rotate h(n) in this way, we do get exactly the expected result (Fig. 1c and Fig. 1d).

This is an empirical fix that we would likely do instinctively upon seeing the problem with Fig. 1a, thus leading us to a correct result, but leaving us to wonder what we had done wrong. But it is not a fix that will work in the even-length case.

3. PURE LINEAR PHASE AND REALISTIC LINEAR PHASE

The problem above was a failure to address the requirement of getting the phase correct. For the most part, we will be looking to achieve a so-called "linear phase." However, in most cases, a true, pure, linear phase is not necessary or even possible. Further, there are in most text books various classes of linear phase, all of which represent perfectly good filters. What we often find on close examination is an attempt to represent the phase as a pure linear phase for the purpose of describing the desired response. This done, we may well find that the actual phase that is achieved is a sub-category of pure linear phase - specifically that the actual phase includes phase jumps of π in the stopbands. This is an entirely expected, desirable, and necessary result, and is in no ways a defect.

Specifically, a pure linear phase would be characterized by

$$e^{j\phi(\omega)} = e^{-j[(N-1)/2]\omega} \tag{7}$$

for a length N FIR filter. Here $\phi(\omega)=-[(N-1)/2]\omega$, hence the linearity of the phase with frequency. This is a constant time delay of [(N-1)/2].

Also associated with the idea of linear phase is a symmetry of the impulse response about some center. This symmetry criterion includes a general class of linear phase response, and provides an extremely useful viewpoint from which we can develop and then understand the overall problem in terms of an amplitude function.

Here suppose that we have an odd length filter (for specificity, N=5) with a symmetric impulse response:

$$H(e^{j\omega}) = h_2 + h_1 e^{-j\omega} + h_0 e^{-2j\omega} + h_1 e^{-3j\omega} + h_2 e^{-4j\omega} \tag{8}$$

Here the subscripts on the h's simply are used to distinguish specific numbers, and are not of course the time indexes. Note that we can rearrange equation (8) as follows (see also Fig. 2a):

$$H(e^{j\omega}) = e^{-2j\omega} [\, h_2 e^{2j\omega} + h_1 e^{j\omega} + h_0 + h_1 e^{-j\omega} + h_2 e^{-2j\omega}]$$

$$= e^{-2j\omega} [\, a_0 + a_1 \cos(\omega) + a_2 \cos(2\omega)\,] \tag{9}$$

where $a_0=h_0$, $a_1=2h_1$, and $a_2=2h_2$.

A similar, but nonetheless different, expansion is available for even length (done here for N=6):

$$H(e^{j\omega}) = h_3 + h_2 e^{-j\omega} + h_1 e^{-2j\omega} + h_1 e^{-3j\omega} + h_2 e^{-4j\omega} + h_3 e^{-5j\omega}$$

$$= e^{-(5/2)j\omega} [a_1 \cos(\omega/2) + a_2 \cos(3\omega/2) + a_3 \cos(5\omega/2)] \quad (10)$$

where $a_1=2h_1$, $a_2=2h_2$, and $a_3=2h_3$. An example showing the components of this amplitude function is seen in Fig. 2b.

The way we interpret these is in terms of a pure linear phase $e^{-j[(N-1)/2]\omega}$ times an "amplitude" function $A(\omega)$ which is the sum of cosines, as indicated. That is, we decompose $H(e^{j\omega})$ into two multiplicative terms:

$$H(e^{j\omega}) = e^{-j[(N-1)/2]\omega} \cdot A(\omega) \quad (11)$$

as compared to a more classic multiplicative decomposition:

$$H(e^{j\omega}) = e^{j\phi(\omega)} \cdot |H(e^{j\omega})| \quad (12)$$

in terms of a phase $\phi(\omega)$ and a magnitude $|H(e^{j\omega})|$. From equations (11) and (12) it is clear that $A(\omega)$ and $|H(e^{j\omega})|$ are not the same thing unless $A(\omega)$ is always positive. This would likely only be true for filters that have no stopbands (perhaps some kind of amplitude equalizer). However, other filters such as common low-pass, band-pass, and high-pass will have bands where we intend to make the response approximate zero. In making it approximate zero, we need to cross zero one or more times, and this corresponds to zeros on the unit circle, and corresponding jumps of π in phase, interrupting any pure linear phase.

Another way to look at it is to write:

$$e^{j\phi(\omega)} |H(e^{j\omega})| = H(e^{j\omega}) = e^{-j[(N-1)/2]\omega} \ \mathrm{sgn}\{A(\omega)\} \ \mathrm{sgn}\{A(\omega)\} \ A(\omega) \quad (13)$$

where $\mathrm{sgn}\{A(\omega)\}$ is the sign of $A(\omega)$, and hence $\mathrm{sgn}\{A(\omega)\}A(\omega) = |H(e^{j\omega})|$ and we then see that the actual phase is:

$$e^{j\phi(\omega)} = e^{-j[(N-1)/2]\omega} \ \mathrm{sgn}\{A(\omega)\} \quad (14)$$

from which we see that the actual phase is the pure linear phase as flipped (hence π phase jumps) by the sign of $A(\omega)$.

To be clear, we want to assign a phase, and we will assign a linear phase [as in equation (7)], but we expect to end up with a phase as in equation (14), which is not a pure linear phase. The importance of these phase jumps of π should not be misunderstood. These represent an inversion of the signal, and this is not the same thing as delaying the signal by half a cycle. While inversion changes the phase at any frequency by 180°, an inversion represents no delay (see Fig. 3). This point can be further emphasized by computing the classical "group delay."

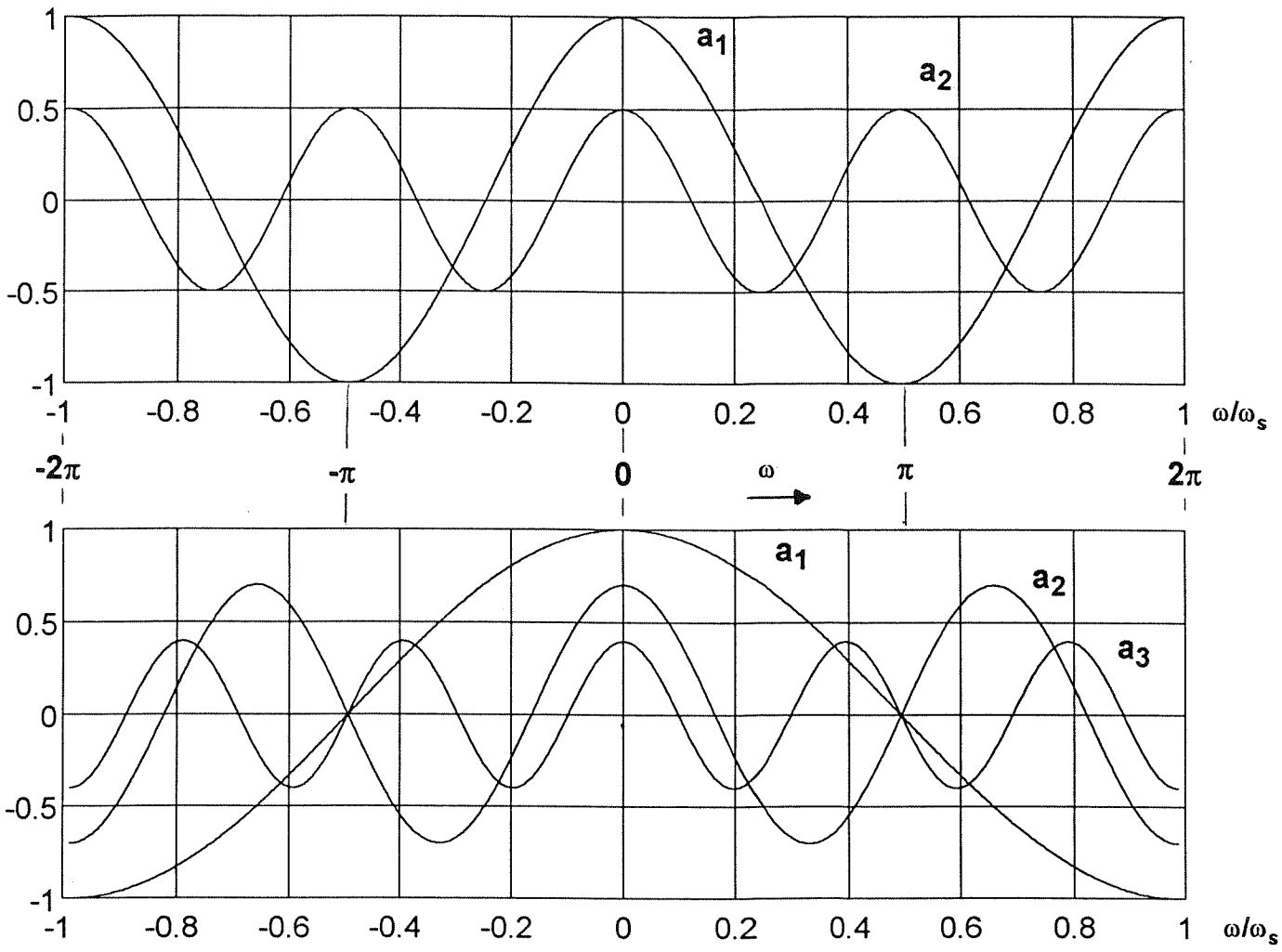


Fig. 2 Amplitude components for odd (top) and even (bottom) length filters

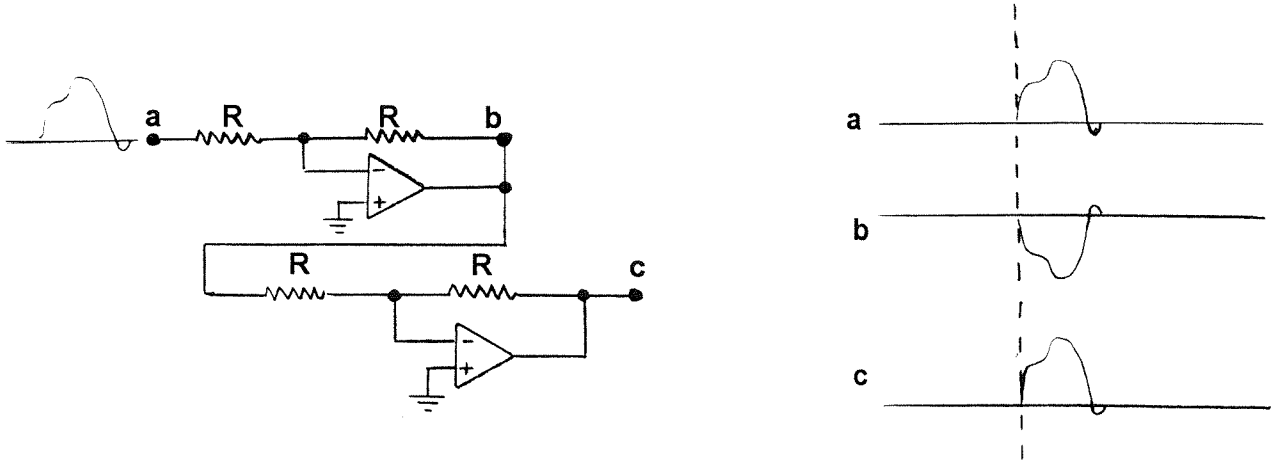


Fig. 3 An inverter does not correspond to a delay. If it did, two inverters would be twice that delay, and yet two inverters in series are no delay. If one inverter were a delay, and the other a compensating advance, which one is which, since both inverters are identical?

4. APPLYING LINEAR PHASE TO INVERSE DFT METHOD - ORDINARY FREQUENCY SAMPLING

Having now looked at the general principles involved, we can think about how to write a frequency sampling program or series of programs. Here we will be looking at a series of programs for different requirements and for different input/output options. It is left to the reader to combine ideas into a single program if that seems useful.

The first program we will look at is fsamp.m which uses the inverse DFT and hence, samples are equally spaced in frequency. This program is capable of internally generating all the individual frequency samples from a less detailed specification in terms of passbands and stopband. It also add a nominal linear phase appropriate for the filter length. Specifically, for this program, a filter specification format (input parameters to fsamp.m) consistent with other MATLAB™[1] filter design functions will be used. [That is, the input parameters to the function fsamp.m will be used to internally generate the actual sample points (frequencies and corresponding amplitudes) instead of having to enter or otherwise generate each point individually.] This of course constrains us to certain filter options, but it is easier for simple filters.

Optionally with fsamp.m, the user can input a vector, m, of amplitude samples, in which case the internally generated default is ignored. This option is perhaps most useful in conjunction with repeated use of fsamp.m while also employing the optional output, mout. The mout output option keeps the last amplitude vector used. In the case of an initial run with no input vector m, mout is the internally generated amplitude vector obtained from the initial specifications. This is very often an extremely useful initial trial vector even when we want to refine it. This allows one to start with an excellent initial guess and then to iteratively trim the amplitude vector. This is commonly useful when we adjust one or more transition band samples (see Examples 3 and 4 below).

The program is straightforward. It begins by generating the frequencies at which samples will be computed, and then generates the corresponding linear phase term at those frequencies. This starts with a pure linear phase, but then inverts the phase for the upper half of the frequency vector in the case of an even number of samples (that is, heading toward an even length filter). This is best understood with reference to Fig. 2. From Fig. 2 we note that the amplitude function for an odd length filter is even symmetric about 0 and about $f_s/2$ (π). The even length filter has an amplitude that is even symmetric about 0 but odd symmetric about $f_s/2$ (π). Another way to look at it is that the even length filter is an expansion in terms of odd multiples of $\omega/2$ and thus has period 4π instead of 2π. Perhaps most simply, we see that there is a zero at $f_s/2$ for the even length case, the well-known "automatic" zero at z=-1 in the z-plane the occurs for even length.

PROGRAM 1: fsamp.m

```
function [h,mout] = fsamp(N,f,a,m)
%   function [h,mout] = fsamp(N,f,a,m)
%
%    N = length
%    f = frequency vector on 0-1 with 1 = half the sampling freq.
%    a = amplitude vector
%    m = optional input amplitude
%    mout outputs the latest amplitude, which is either generated
%      by the first run of the program, or provided by the
%      optional input m.  mout can be modified externally and
%      then used as an input for another iteration.
%    The alternative input amplitude m can be used to override
%      the default two-band filter if desired
%
%    B. Hutchins                                    Fall 1995
%

k=0:N-1
w=2*pi*k/N;
%-------------------generate phase-----------------
ph=exp(-j*w*(N-1)/2);
if mod(N,2)==0;
  ps=[ones(1,N/2) -ones(1,N/2)];
  ph=ph.*ps;
end

%-------------generate default amplitude------------
for n=1:N
       if w(n) < f(2)*pi; mask(n)=a(2);
          elseif w(n) > (2*pi-f(2)*pi); mask(n)=a(2);
          else mask(n)=a(3);
       end
end
%------replace by specified amplitude if provided-----
if exist('m')==1; mask=m; end

H=mask.*ph;

h=ifft(H);
figure(1);
subplot(221)
stem(k,h);
axis([-2 N+2 -.3 .7]);
MH=abs(freqz(h,1,500));
MH=MH/MH(1);
subplot(222);plot([0:.001:.499],MH)
axis([-.05 .55 -.3 1.3]);
grid
MHDB=db(MH,120,20);
MHDB=MHDB-MHDB(1);
subplot(224);plot([0:0.001:.499],MHDB)
axis([-.05 .55 -50 10]);
grid
figure(1)
% output latest mask
mout=mask;
```

Now, in the case of frequency sampling, as a consequence of wanting to use the DFT, and of the way the DFT is defined, we are forced to choose frequencies from 0, up through $f_s/2$ (π), to just one sample short of f_s (2π). Thus we cross the

break at $f_s/2$ (π) and must deal with the odd symmetry in the even length case. We did not need to deal with this in the case of weighted integrated square error (AN-332) because we integrated from $-\pi$ to π, and not from 0 to 2π. In total, this probably amounts to little more than an interesting quirk.

Having now generated the correct phase at the correct frequencies, we next will generate the corresponding amplitudes. This we do by comparing each of the frequencies with the band-edges specified on the command line. This is quite analogous to the manner in which we arrived at the samples in equation (2). Here we have written code that is restricted to a two band filter, but modification for more bands is not at all difficult. Having now generated this vector of amplitude samples (called here the mask), we will just discard it if there is present an optional input vector m in the command statement. Whichever mask remains, the default or the input, is then multiplied by the phase to give the actual frequency samples H.

This completes the design with the exception of the key step of obtaining the impulse response as the inverse DFT of the frequency samples H. The remainder of the program deals with plotting.

Below we give same examples illustrating the use of this first program. Example 1 (Fig. 4) is fundamental and entirely analogous to the case of Fig. 1. This is generated by:

```
[h,mout] = fsamp( 19, 2*[0 0.25 0.25 0.5], [1 1 0 0] )          (15)
```

and thus calls for a length 19 low-pass filter with a cutoff frequency of $f_s/4$. Here the frequency vector is written in terms of bandedges; with a passband from 0 to $0.25f_s$, and a stopband from $0.25f_s$ to $0.5f_s$. This vector is multiplied by 2 here since we are employing the MATLAB filter specification standard where $f_s=2$. The amplitude vector [1 1 0 0] here describes the amplitudes at the ends of the bands.

The result is very typical of frequency-sampling design, and quite similar to most FIR low-pass designs. From Fig. 4a, we find a sinc-like, length-19 impulse response, symmetric about the center. The magnitude response, Fig. 4b is seen to pass through 1 at five equally spaced points, and then through 0 at five equally spaced points. The plot in db, Fig. 4d can be seen to have a sidelobe rejection of about 16db. Finally, the plot of the zeros of h(n) (Fig. 4c) is typical in having zeros on the unit circle in the stopband, and zeros in reciprocal-complex-conjugate quadruples in the passband. Note that each dip in the passband thus corresponds to four zeros total. Each null in the stopband corresponds to two (complex conjugate) zeros. This filter, being of odd length, has no zero at z=-1.

Example 2 shown in Fig. 5 is fairly similar, except it shows a case that is a somewhat longer length, and an even length. The zero plot is not shown, but is

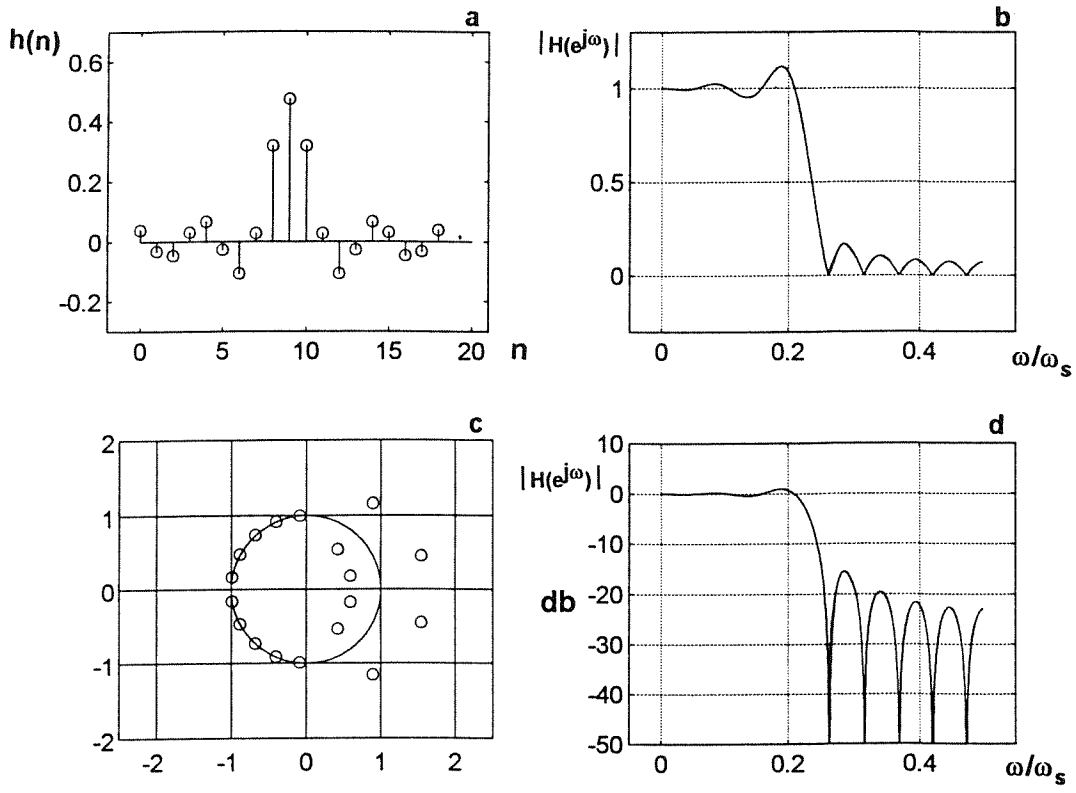


Fig. 4 Example 1: A basic length-19 low-pass as an example of odd length

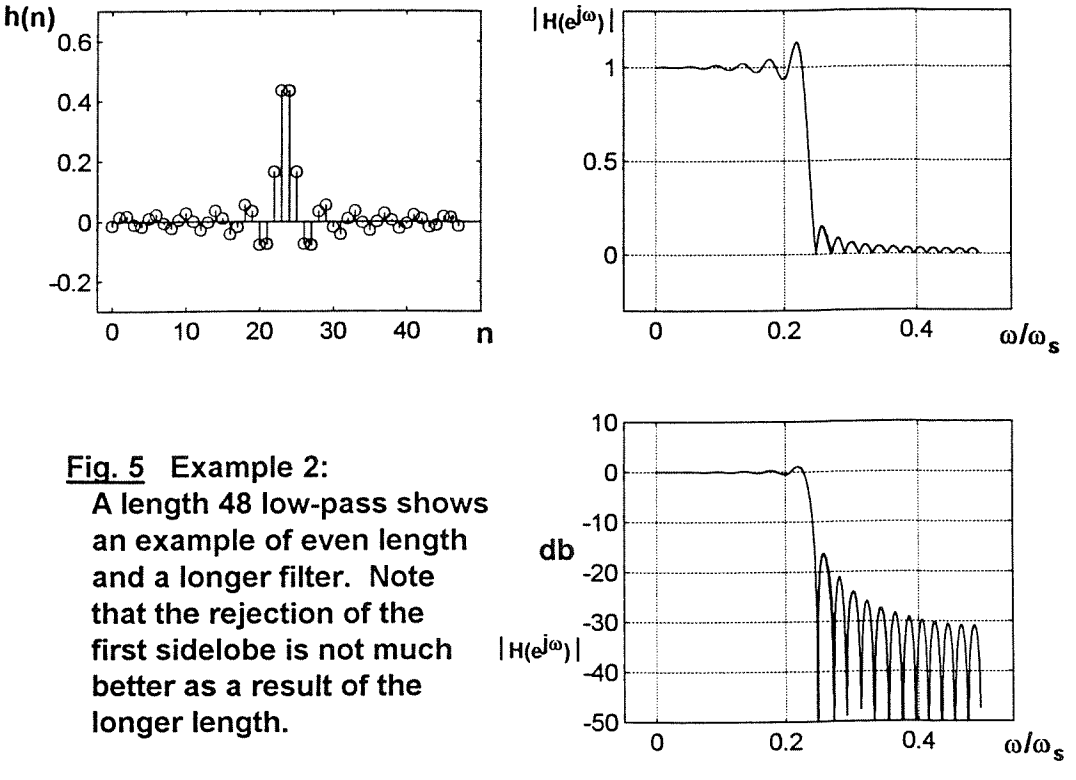


Fig. 5 Example 2:
A length 48 low-pass shows an example of even length and a longer filter. Note that the rejection of the first sidelobe is not much better as a result of the longer length.

similar to the first example except it has more zeros, and this case has a zero at z=-1. Even without the plot we know that there is a zero at z=-1 since we see the magnitude plots go down at $0.5f_s$ rather than remain high as Example 1 did. Most significantly, note that while the cutoff is sharper, the ripple in the passband and stopbands is not much different than the length-19 case (again, about -16db for the first sidelobe). Thus it is seen that a longer length will not improve stopband rejection, except as the sidelobes roll-off faster.

At this point we will turn our attentions to some manipulation of the samples to see what improvement in stopband rejection might be achieved thereby. This particular program has been written to make this fairly easy to do. Here we will start with the case of Example 1, and use its output vector:

mout = [1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1] (16)

and modify it as:

m = [1 1 1 1 .5 0 0 0 0 0 0 0 0 0 .5 1 1 1] (17)

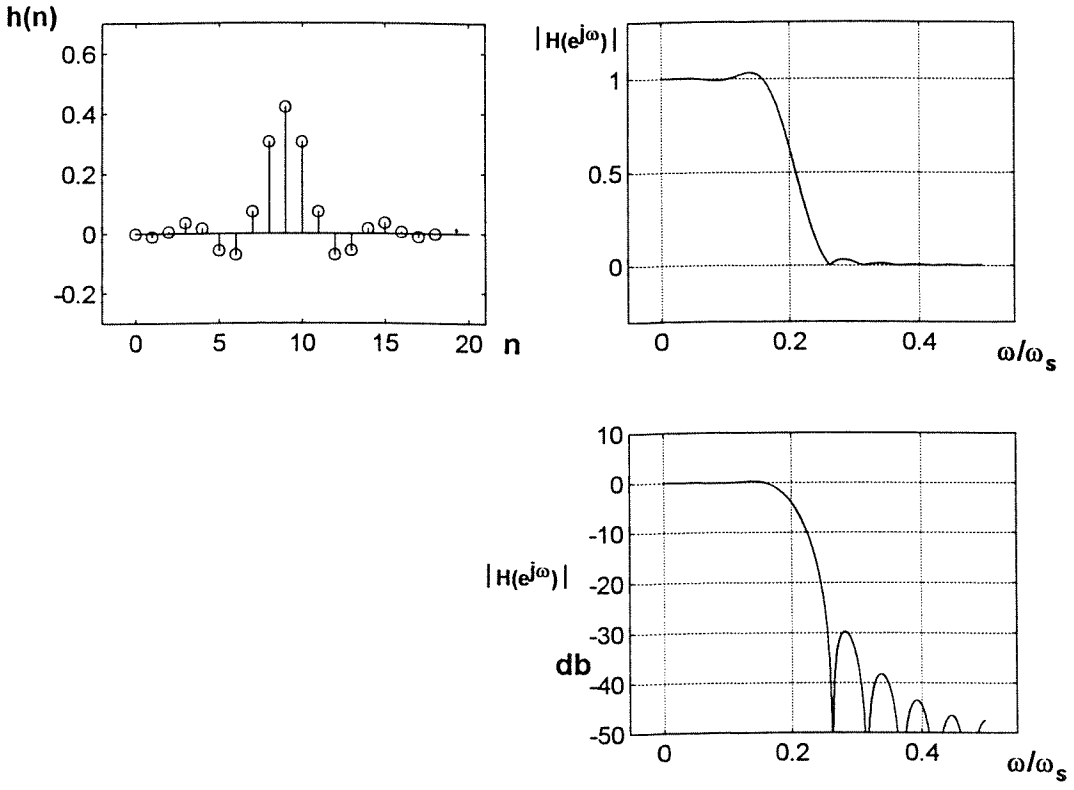


Fig. 6 Example 3: Length 19 with transition-band sample set to 0.5 results in less sharp cutoff, but greater sidelobe rejection.

we can then simply add this to the input command line:

$$[h,mout] = fsamp(19, 2*[0\ .25\ .25\ .5], [1\ 1\ 0\ 0], m) \qquad (18)$$

This result, shown in Fig. 6 (Example 3) has a less sharp cutoff compared to Example 1, but a significantly better stopband rejection (about -30db). In one more iteration, Fig. 7 (Example 4), we modify mout of Example 3 (same as m at the input of Example 3) for a new m as:

$$m = [1\ 1\ 1\ .85\ .5\ .15\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ .15\ .5\ .85\ 1\ 1] \qquad (19)$$

The result is seen in Fig. 7 where we see an even less sharp cutoff and an even better stopband rejection (about -43 db).

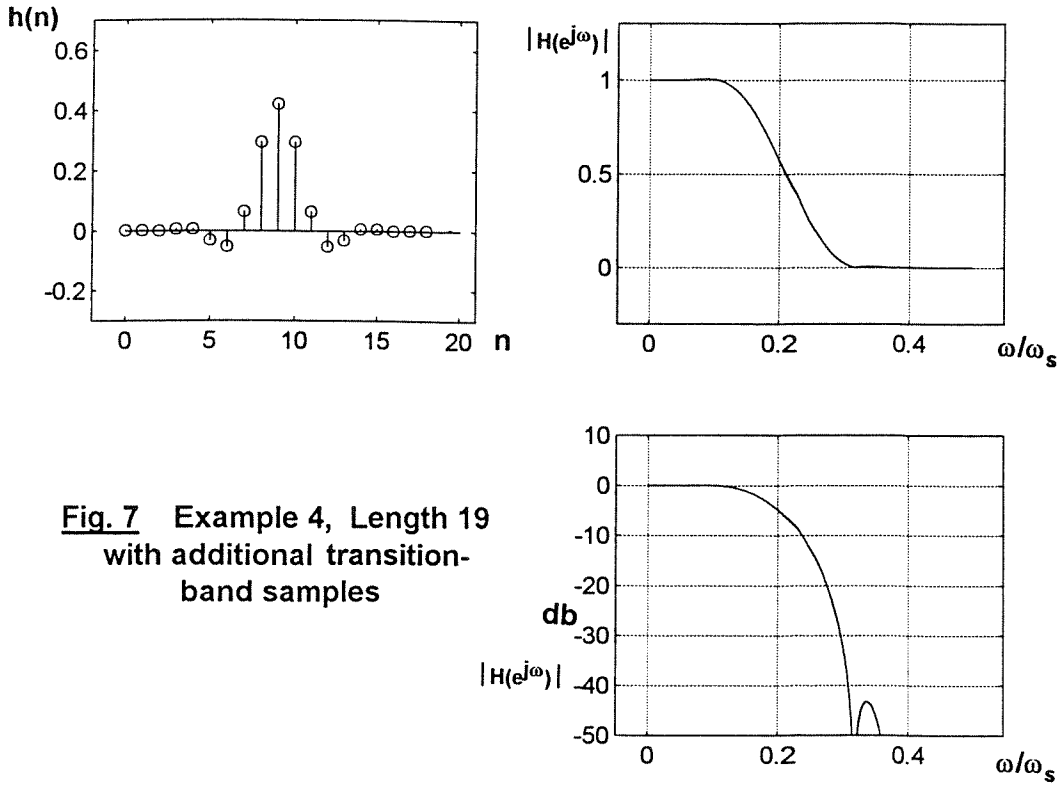


Fig. 7 Example 4, Length 19 with additional transition-band samples

5. UNEQUAL SPACING OF FREQUENCY SAMPLES

Frequency sampling in a more general sense need not require equally spaced samples. Instead, we seek a length N impulse response h(n) that is related to N frequency samples by N equations in N unknowns. If we set up and solve this problem, one of its special cases will be equally spaced samples and a matrix representing the N×N equations that is the same as the DFT matrix.

Before setting up the equations leading to the program gfs.m, we should mention two important consequences of using unequal spacing. First, there will be no automatic way of generating the samples from specified passbands. Instead, we will in general need an input vector f of frequencies and a corresponding vector of amplitudes. Secondly, we need to recognize that there can be severe consequences of choosing the unequal spaced points.

The basis for the program here goes back to equation (2). We simply have in mind N frequencies ω_k with N corresponding versions of equation (2), thus giving us our N equations with N knowns (H), and N unknowns (h). While this works for any N frequencies and corresponding samples, we will be looking here for choices of sample values that give us real and symmetric (linear phase) impulse responses. The same phase inversion for samples from π to 2π that was required for even length in fsamp.m is also needed for even length here.

As an example, if we have N=4, equation (2) yields four equations:

$$H(e^{j\omega 1}) = h_0 + h_1 e^{-j\omega 1} + h_2 e^{-2j\omega 1} + h_3 e^{-3j\omega 1} \tag{20a}$$

$$H(e^{j\omega 2}) = h_0 + h_1 e^{-j\omega 2} + h_2 e^{-2j\omega 2} + h_3 e^{-3j\omega 2} \tag{20b}$$

$$H(e^{j\omega 3}) = h_0 + h_1 e^{-j\omega 3} + h_2 e^{-2j\omega 3} + h_3 e^{-3j\omega 3} \tag{20c}$$

$$H(e^{j\omega 4}) = h_0 + h_1 e^{-j\omega 4} + h_2 e^{-2j\omega 4} + h_3 e^{-3j\omega 4} \tag{20d}$$

which are in matrix form:

$$H = Mh \tag{21}$$

where

$$M(k,n) = e^{-jn\omega k} \tag{22}$$

[In inverting equation (21) with MATLAB, we need to recognize that to get the required:

$$h = M^{-1}H \tag{23}$$

we need to use the transpose of H, and in MATLAB notation, this is H.' and not just H', which is always the conjugate transpose.] Note that if the frequencies are equally spaced we would have $\omega_k = (2\pi/N)k$ and the matrix M becomes:

$$M(k,n) = e^{-j(2\pi/N)nk} \tag{24}$$

which is the DFT matrix.

The program gfs.m allows experimentation with unequal spacing, although in many cases, we will find it more convenient to work with the amplitude function (program amp.m below).

Example 5 (Fig. 8) shows the unequal spacing program used in an equal spacing application. The command lines are:

```
f = [0 .05 .1 .15 .2 .25 .3 .35 .4 .45 .5 .55 .6 .65 .7 .75 .8 .85 .9 .95]      (25a)

a = [1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1]                                  (25b)

h = gfs(f,a)                                                                   (25c)
```

The result is the length 20 impulse response as shown in Fig. 8. The corresponding magnitude response shows the amplitude values as specified.

Next, in Example 6 (Fig. 9), we will get around to an unequal spacing of samples. The simplest case will be to move one sample. Instead of choosing a sample at 0.15, we will move it to 0.17. This also means that we need to move the sample at 0.85 down to 0.83. Thus we have as the only change:

```
f = [0 .05 .1 .17 .2 .25 .3 .35 .4 .45 .5 .55 .6 .65 .7 .75 .8 .83 .9 .95]      (26)
```

The impulse response for Example 6 is seen in Fig. 9a, and is virtually identical to Fig. 8a on the scale shown, indicating right away that nothing drastic is happening here. Fig. 9b shows the corresponding magnitude response, again very much like Fig. 8b, but with the one difference that the response now goes through 1 at 0.17, not at 0.15, the one change that we specified. This shows that the method does work as we desired.

Example 7 (Fig. 10) shows another application of the unequal spacing program, this time to an equally spaced case, but with an initial offset. The command lines are:

```
f = [ .5 .15 .25 .35 .45 .55 .65 .75 .85 .95 ]                                 (27a)

a = [ 1  1  1  0  0  0  0  1  1  1 ]                                          (27b)

h=gfs(f,a)                                                                     (27c)
```

Here note that for the first time, we see that there are the same number of non-zero samples at the high end of the vector a as there are at the low end (previously, there was always one fewer on the high side). This is because we have chosen samples at intervals of 0.1, but offset by 0.05. Note that the frequency sampling still works fine. We do get an extra zero at 0.5 due to the even length (once again, consider Fig. 2b).

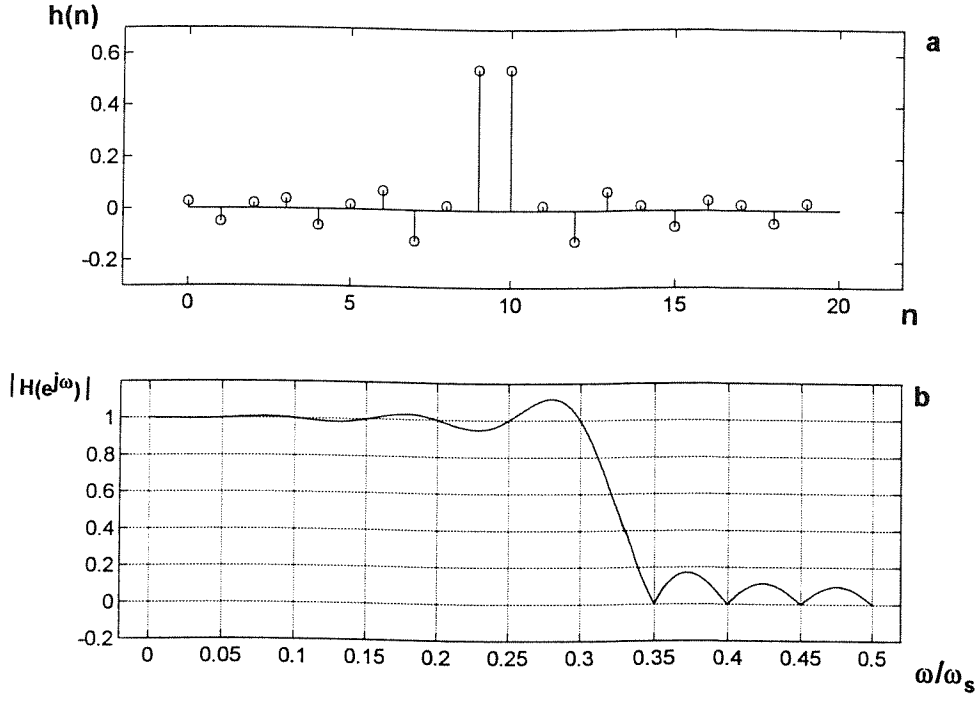


Fig. 8 Example 5: Unequal spacing program applied to equal spacing

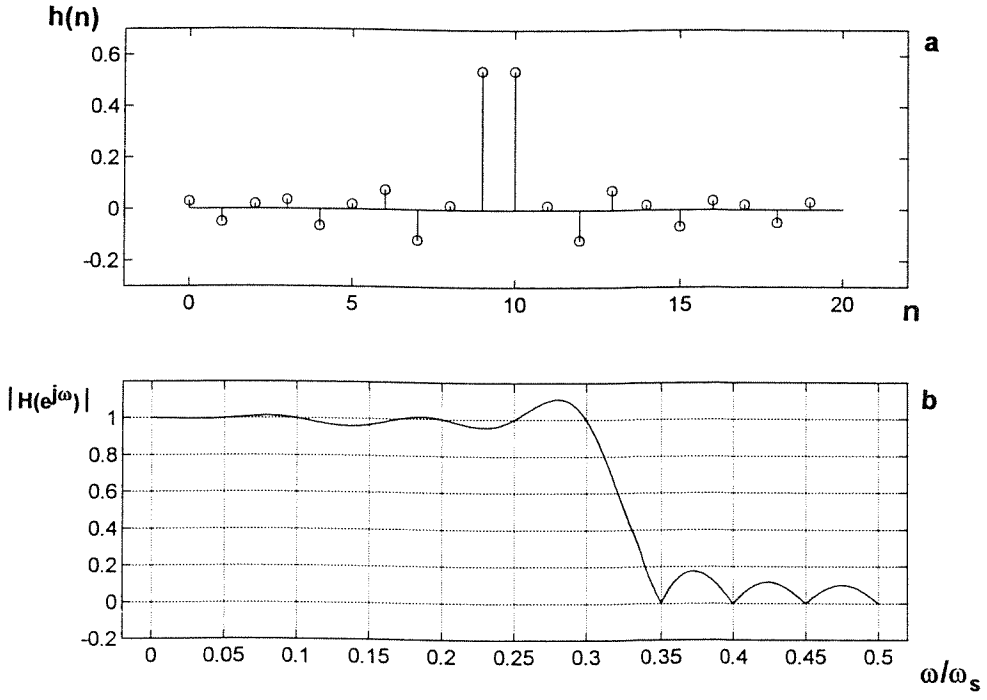


Fig. 9 Example 6: Sample at 0.15 (Example 5) moved to 0.17 here

PROGRAM 2: gfs.m

```
function h=gfs(f,a)
%
%               GENERALIZED FREQUENCY SAMPLING
%         even/odd order  regular/irregular spacing
%                    linear phase assumed
%
%   h = gfs(f,a)
%        f = frequency vector on 0 to fs=1
%        a = corresponding amplitudes
%
%   B. Hutchins                                Jan 1996

N=length(f);
w=2*pi*f;
%   - linear phase -
ph=exp(-j*w*(N-1)/2);
%   - odd length amplitude odd symmetry about fs/2 -
if floor(N/2)==(N/2)
     ph = ph.*[ones(1,N/2) -ones(1,N/2)];
end
H=a.*ph;

for n=1:N
     for k=1:N
          M(k,n)=exp(-j*(n-1)*w(k));
     end
end

h=inv(M)*H.';

k=0:(length(h)-1);
figure(1);
subplot(211)
stem(k,h);
MH=abs(freqz(h,1,500));
MH=MH/MH(1);
subplot(212);plot([0:.001:.499],MH)
grid
```

6. SAMPLING THE AMPLITUDE FUNCTION

As mentioned with regard to the gfs.m program, it is frequently easier and more computationally efficient to work with the sampling of the amplitude function rather than with a sampling of the frequency response itself. With this approach, we can automatically includes the linear phase. Also, we only need to specify the lower half of the response: 0 to π. In addition, we are in general then only solving half as many equations with half as many unknowns, meaning an inversion of the matrix of only about 1/4 the original size.

Samples may be chosen for the amplitude function for the odd length case:

$$A(\omega_k) = a_0 + a_1 \cos(\omega_k) + a_2 \cos(2\omega_k) + \ldots \qquad (28)$$

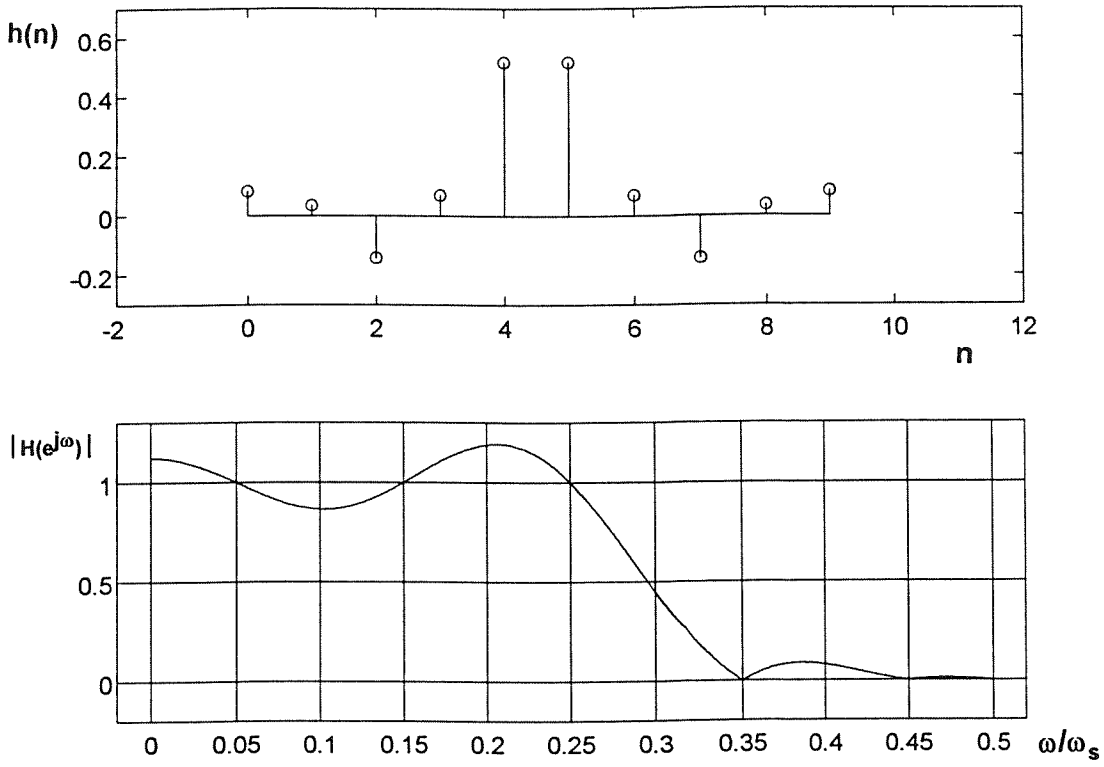


Fig. 10 Example 7: Equal spacing offset by 0.05

while for even length, the amplitude function is:

$$A(\omega_k) = a_1 \cos(\omega_k/2) + a_2 \cos(3\omega_k/2) + a_3 \cos(5\omega_k/2) + \ldots \quad (29)$$

Thus L samples can result in either a length 2L-1 odd length filter or a length 2L even length filter. The extra tap in the even length filter is merely a consequence of the automatic zero at z=-1 that occurs with even length.

The program amp.m actually designs both an even length and an odd length option by setting up L equations in L unknowns using equation (28) or equation (29). Here the matrix elements are obtained by evaluating cosines at the specified frequencies, but otherwise, the entire program is quite similar to gfs.m.

Here we are again looking at the issue of unequal spacing of frequency samples, and want to look at a wider range of examples. However, it is useful to begin with examples related to equal spacing as a means of better understanding the sampling of amplitude. That is, it is useful to look at conventional-looking responses before looking at the more unusual ones that result when we exercise the flexibility of the spacing options.

PROGRAM 3: amp.m

```
function [ho,he]=amp(f,a)


%   h = amp(f,a)
%        f = frequency vector on 0 to fs/2 (fs/2 = 1/2)
%        a = corresponding amplitudes
%
%   Does both even and odd length filters based on amplitude
%
%   B. Hutchins                Jan 1996

L=length(f);
w=2*pi*f;

% Matrix for even length
  for n=1:L
     for k=1:L
          ME(k,n)=cos((n-1/2)*w(k));
     end
  end

% Matrix for odd length
  for n=1:L
     for k=1:L
          MO(k,n)=cos((n-1)*w(k));
     end
  end


aae=inv(ME)*a.';
aao=inv(MO)*a.';

% even length case
  for n=1:L
     he(n)=aae(n)/2;
  end
  he=[he(L:-1:1),he(1:L)];


% odd length case
  for n=2:L
       ho(n)=aao(n)/2;
  end
  ho=[ho(L:-1:2),aao(1),ho(2:L)];

% plot odd length result
k=0:(length(ho)-1);
figure(1);
subplot(211)
stem(k,ho);
MHO=abs(freqz(ho,1,500));
MHO=MHO/MHO(1);
subplot(212);plot([0:.001:.499],MHO)
grid
pause

% plot even length result
k=0:(length(he)-1);
figure(2);
subplot(211)
stem(k,he);
MHE=abs(freqz(he,1,500));
MHE=MHE/MHE(1);
subplot(212);plot([0:.001:.499],MHE)
grid
```

Here we will begin with six examples, three runs of amp.m, each of which produces two filter designs (odd and even length). These filters are very short so as to better illustrate the consequences of amplitude sampling.

Figure 11, Examples 8a and 8b show the odd-length and even-length filters that result from a sampling at three frequencies on 0 to 0.5. The three frequencies are 0, 0.2, and 0.4, and the corresponding amplitude samples are 1, 0, and 0. Thus we run:

```
[ho,he] = amp( [ 0  0.2  0.4 ], [ 1 0 0 ] )          (30)
```

After running this, we discover immediately an "old friend," a five tap "moving average" filter for the case of odd length-5, the ho output. This results from the effective equal spacing [see equation (32)]. In fact, this exact same result can be obtained by using either of the previous programs, as:

```
h = fsamp( 5, [ 0  0.02  0.02  1 ],[ 1  1 0 0 ] )    (31)

h = gfs( [ 0 0.2 0.4 0.6 0.8 ] , [ 1 0 0 0 0 ] )     (32)
```

Here, comparing equation (30) with equation (32) is probably the best indication of the savings of amp.m as compared to gfs.m.

The alternative length-6 output he (Example 8b) is also interesting. Note that in going from length-5 to length-6 we must have a zero at z=-1, corresponding to a multiplicative numerator term of (z+1). The term z+1 is a two-tap moving average, and note that the impulse response of Example 8b is indeed a convolution of a 5-tap moving average with a two-tap moving average. In fact, we see that the only difference between Example 8a and Example 8b is the extra zero at z=-1. This of course results in some improvement to the stopband for the highest frequencies. For the sample spacing used here, we note that the odd-length filter is the most familiar.

Moving on to Fig. 12, Example 9a and 9b, we see a second run of the amp.m program, this time as:

```
[ho,he] = amp( [ 0  1/6  2/6 ], [ 1  0  0 ] )       (33)
```

Example 9a, the odd-length case is the curious case here. The result is not low-pass, but more of a band-reject, if anything. Another way to look at it is that it is a three-tap moving average filter with twice the usual delay, and this is easily seen in the lowpass from 0 to 0.25, reflected from 0.25 to 0.5. Note that with the taps on either side of the center going to zero, this amplitude response is of the form $\alpha + \beta\cos(2\omega)$. The peaking around 0.5 can be attributed to the failure to take a sample there.

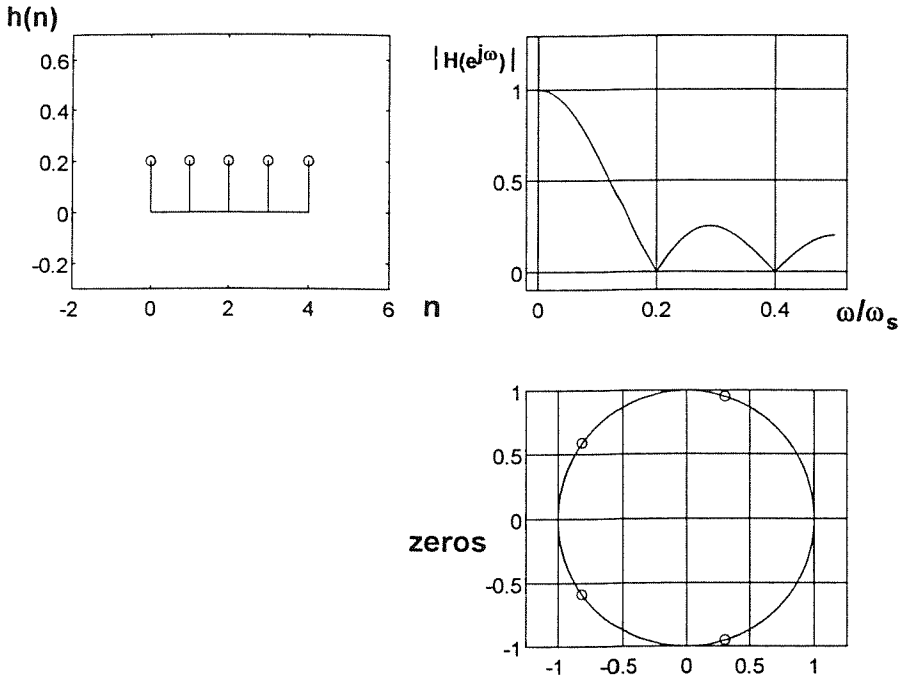


Fig. 11a Example 8a: Samples of amplitude function at 0, 1/5, and 2/5 yield moving average for odd length case.

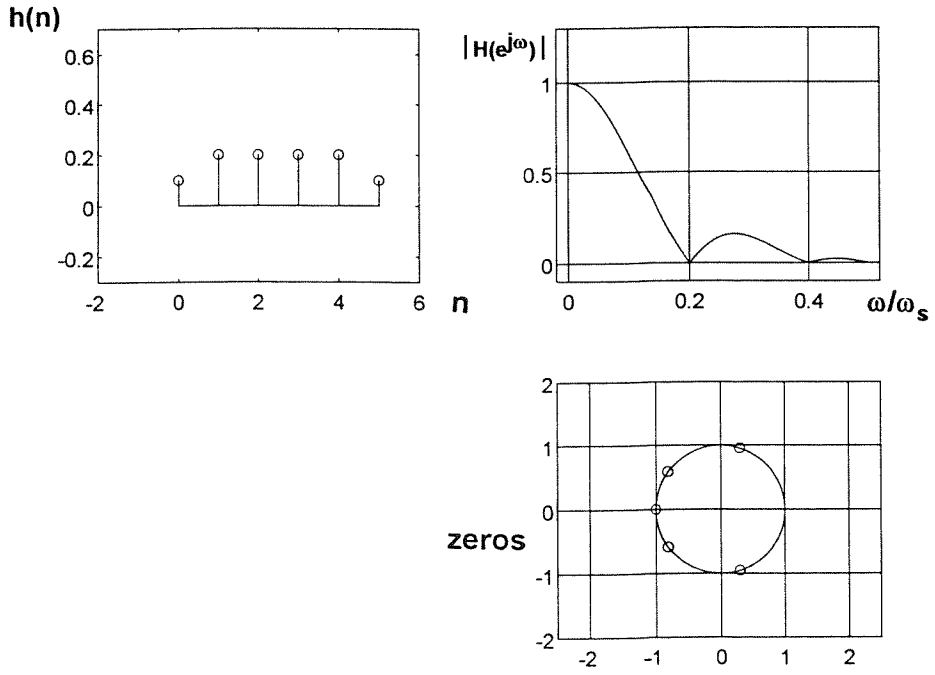


Fig. 11b Example 8b: Samples of Example 8a giving even length filter

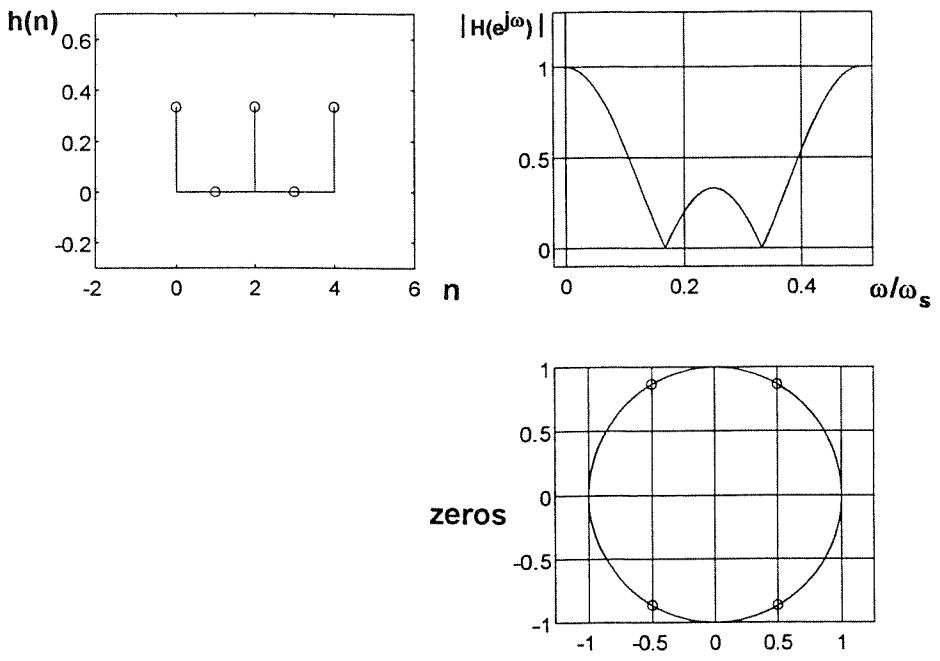


Fig. 12a Example 9a: Samples at 0, 1/6, and 2/6 for odd length case

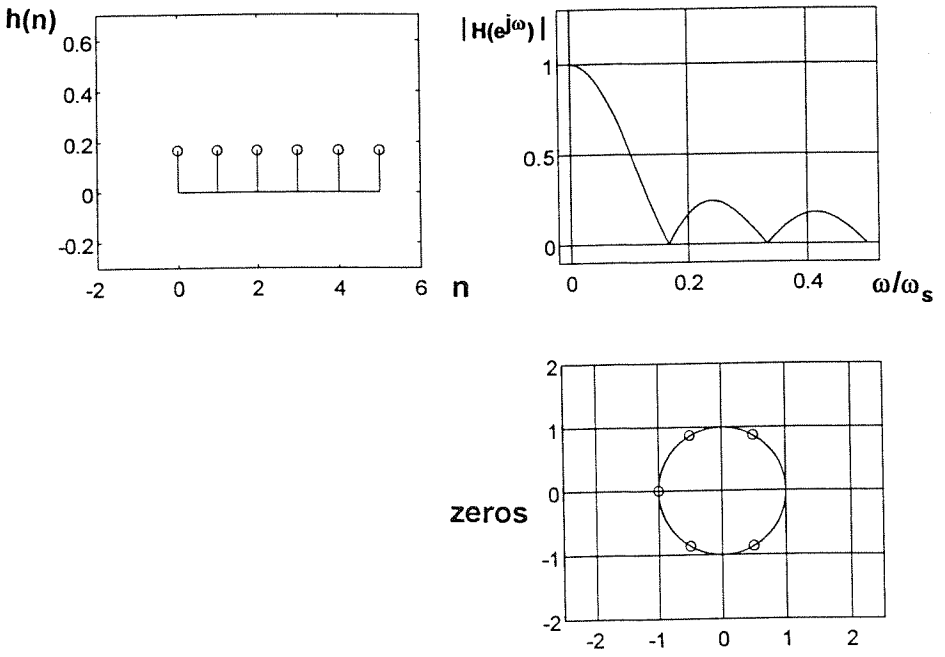


Fig. 12b Example 9b: Samples of Example 9a, even length case

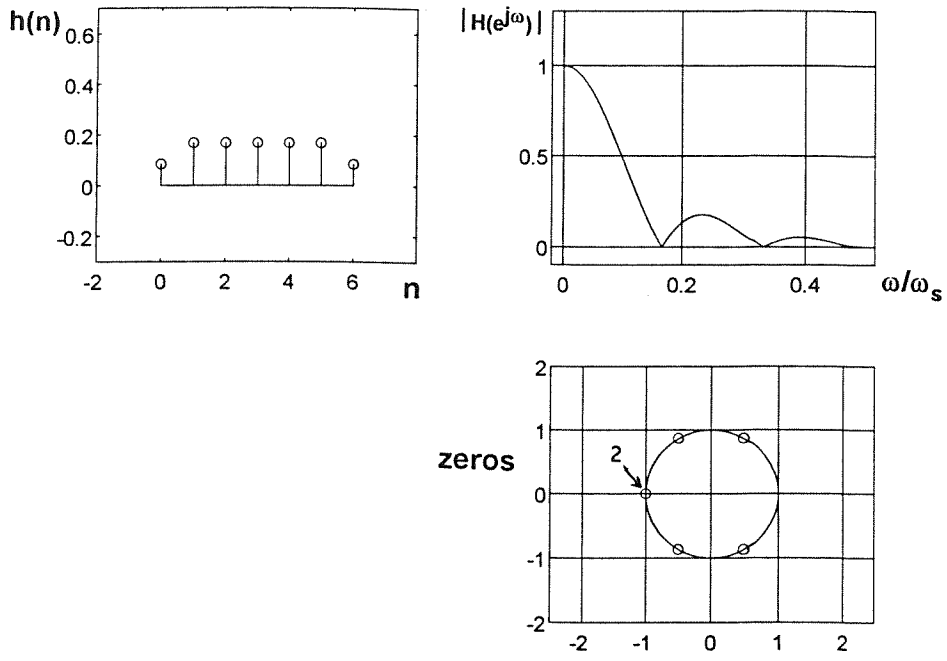


Fig. 13a Example 10a: Adding a sample at $0.5f_s$, odd length case

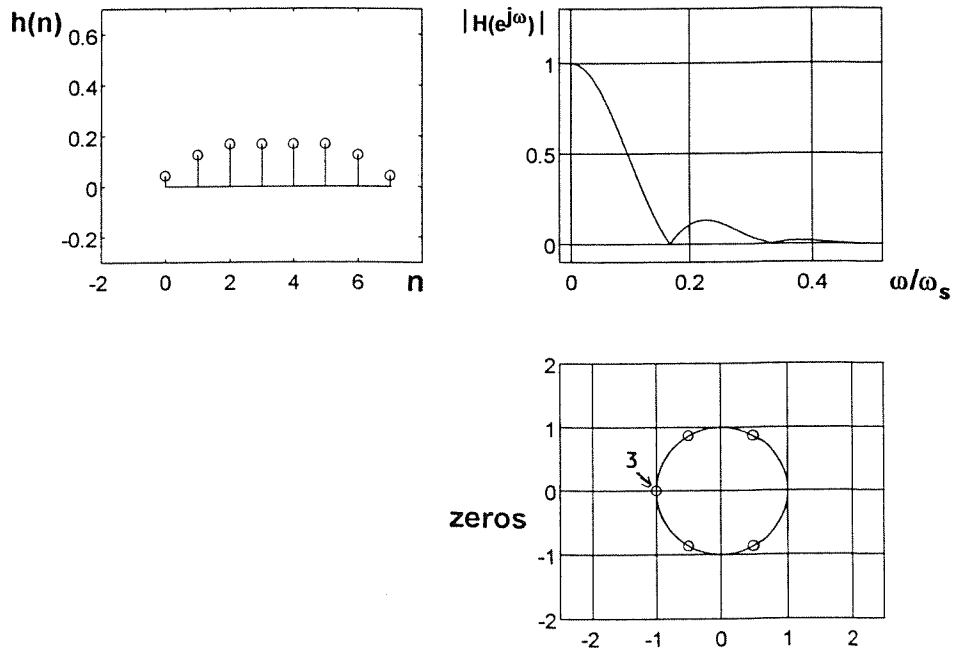


Fig. 13b Example 10b: Adding a sample at $0.5f_s$, odd length case

Example 9b of Fig. 12, the corresponding even length case now is the one that makes sense, coming out as a 6-tap moving average filter. Note that the spacing for this case (1/6) is just what one would use for a length-6 filter. This point can be further emphasized by noting that the exact same 6-tap moving average results from the previous programs as:

```
h = fsamp( 6, [ 0  0.02  0.02  1], [ 1 1 0 0 ] )          (34)

h = gfs( [ 0 1/6 2/6 3/6 4/6 5/6], [ 1 0 0 0 0 0 ] )      (35)
```

Note that the zero pattern of Example 9b is the same as that of Fig. 9a, except here we do have an extra zero, the one at z=-1 that come with even length. Once again we see that the impulse response of the even length filter (Example 9b) is just that of Example 9a, convolved with (z+1).

To this point, with the program amp.m, we have not placed any samples at half the sampling frequency. In one more run, Examples 10a and 10b, Fig. 13, we have a sample specified at $f_s/2$:

```
[ho,he] = amp( [ 0  1/6  2/6  3/6 ], [1 0 0 0] )          (36)
```

This extra amplitude sample, set to zero here, will add two to the lengths of the filters as compared to Examples 9a and 9b. Also associated with this sample at 0.5 is a double zero at z=-1, since the specification on half the band represents both the lower specified band and the upper, reflected, unspecified band (thus including 0.5 twice). Example 10b now uses the same samples as Example 10a, except here we go to an even-length case, and pick up yet one more zero at z=-1, for a total of 3. Note the interesting sequence of Examples 9a, 9b, 10a, and 10b, which correspond to 0, 1, 2, and 3 zeros at z=-1 respectively, with the other four zeros the same in all four cases. In this sequence, we see increasing levels of rejection in the vicinity of 0.5 as the number of zeros at z=-1 increases, as we would expect.

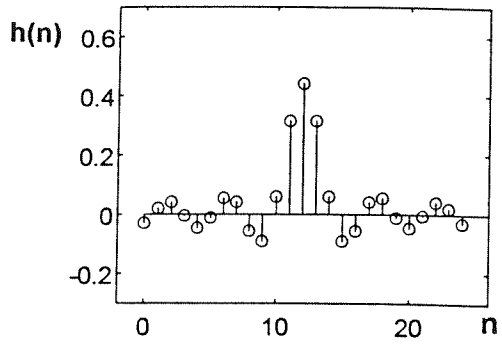
The simple cases in Examples 8, 9, and 10 were presented in an effort to make clear how frequency sampling on the amplitude function works. However, the technique also works for longer, more practical filters, and we will take a look at several of these.

Example 11a, Fig. 14a shows a normal type of equally spaced low-pass, calculated as follows:

```
f = [ 0:0.04:0.48 ]                                       (37)

a = [ 1  1  1  1  1  1  0  0  0  0  0  0 ]                (38)

[ho,he]=amp(f,a)                                          (39)
```



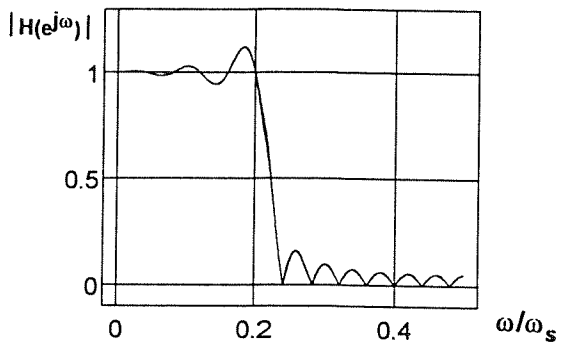
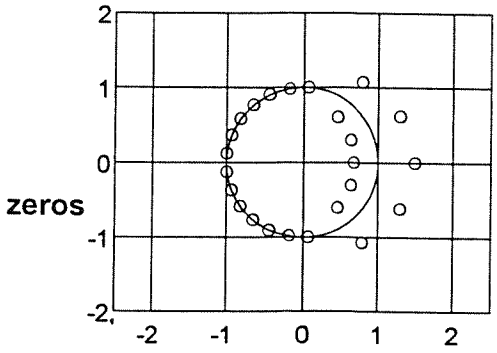
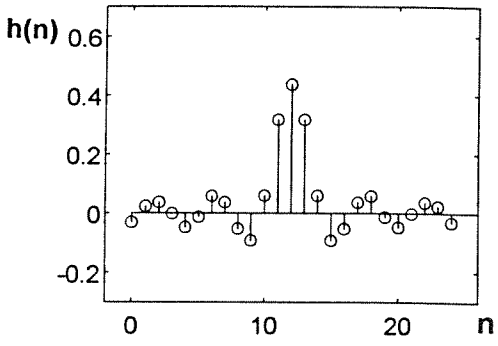


Fig. 14a Example 11a:
A length 25 filter with one sample at $0.4f_s$





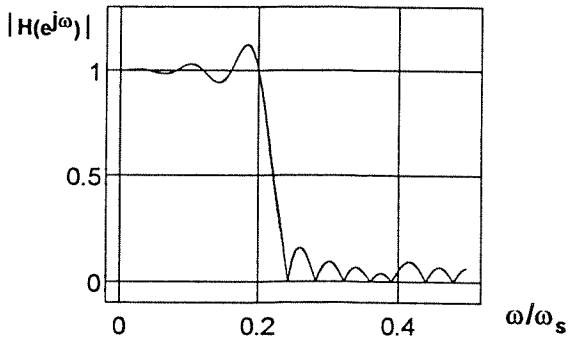
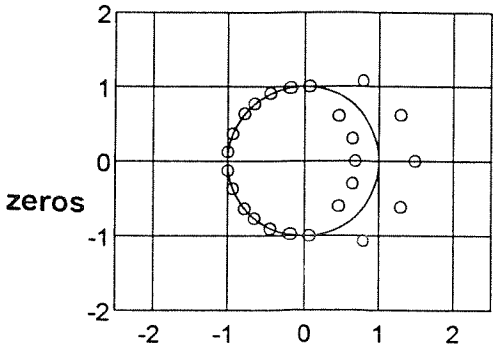


Fig. 14b Example 11b:
A length 25 filter with one sample moved to $0.39f_s$



where we have looked at the odd length option, ho, a length 25 filter. We can perturb the result slightly by moving one sample, much as we did in Example 6 for the passband. Here, we will take the sample that occurs at 0.4 in Example 11a and move it to 0.39 for Example 11b by the following change of one command line:

```
f = [ 0 .04 .08 .12 .16 .20 .24 .28 .32 .36 .39 .44 .48 ]          (40)
```

with Equations (38) and (39) the same. We find now a very similar filter, with the exception that the stopband is slightly rearranged. This filter would be an advantage if we know, for example, that we could expect a strong interfering component at 0.39fs. Such a component would be completely rejected by Example 11b, but not by Example 11a.

In the previous Examples 5 and 6, and here in Examples 11a and 11b, we have taken an existing sample and moved it relatively little to a new, more advantageous position. Another approach which we might consider would be to keep the set of existing, equally spaced samples, and just add another sample to the set. We may be surprised to find that while this does of course cause the response to be completely determined at the added point, that there can also be drastic changes in the rest of the response.

For Example 12, Fig. 15, we use the command lines:

```
f = [0 .04 .08 .12 .16 .20 .24 .28 .32 .36 .39 .40 .44 .48]          (41)

a = [1  1  1  1  1  1  0  0  0  0  0  0  0  0 ]                     (42)

[ho,he] = amp(f,a)                                                  (43)
```

from which we look at the odd case, length 27. (Adding one sample instead of moving an existing one adds two to the length.) We see here that a wide region in the vicinity of the two close samples at 0.39 and 0.40 is now forced to be small. This is perhaps an attractive change in the stopband, but note that the passband ripple increases. The lesson to keep in mind is that two close samples defined to have the same amplitude can have a profound effect on flattening in the general region. This may be good or bad. We certainly need to be aware of it.

Yet another way to use unequal spacing is to use different spacing over different bands of frequencies. This can work in a manner similar to weighting different bands with different levels of importance. It can also be full of surprises. Example 13 (Fig. 16) is entered as:

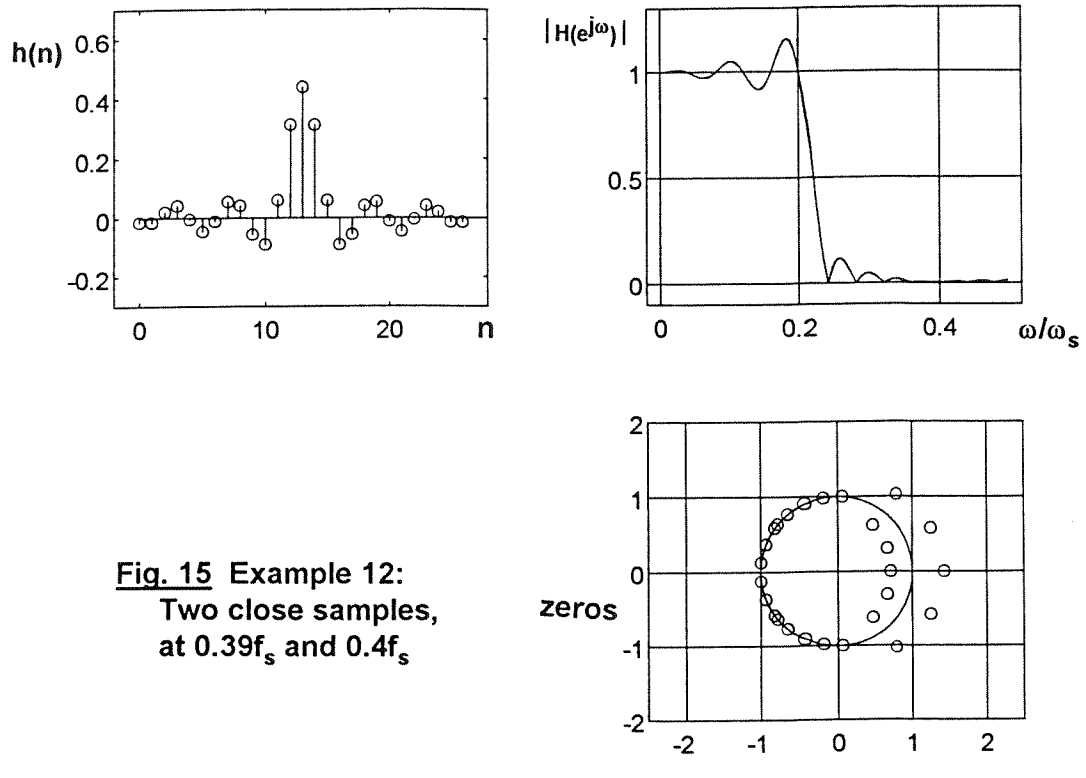


Fig. 15 Example 12:
Two close samples,
at $0.39f_s$ and $0.4f_s$

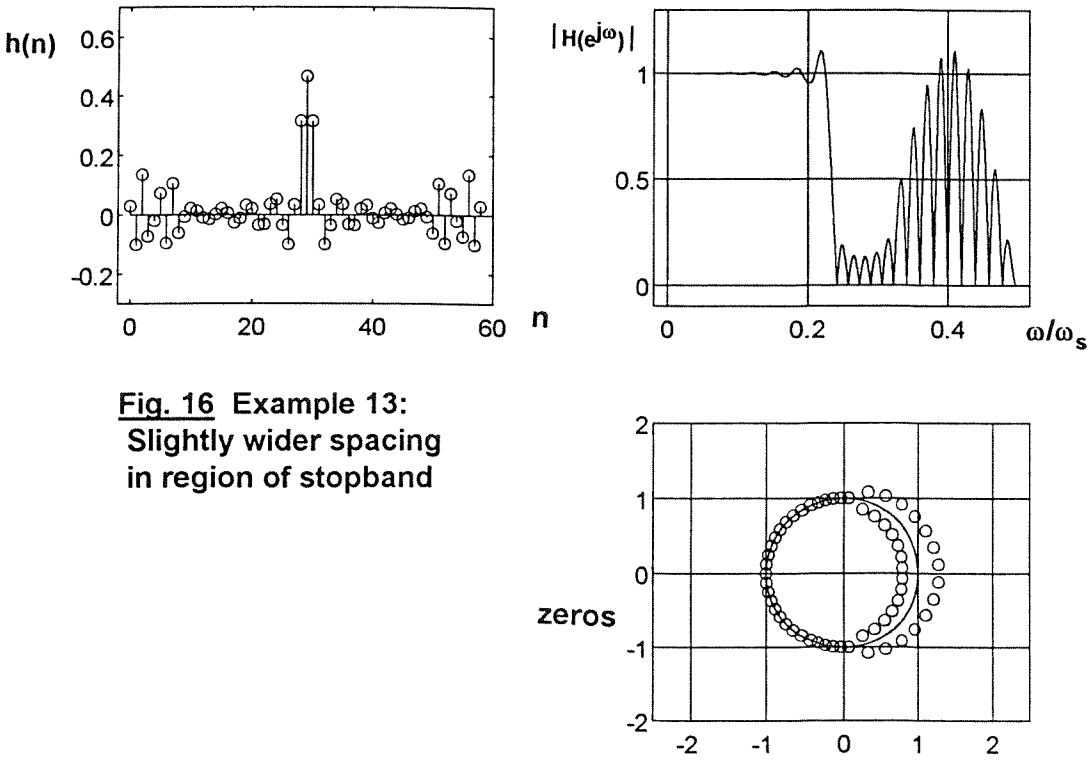


Fig. 16 Example 13:
Slightly wider spacing
in region of stopband

```
f = [ 0:0.016:0.32,  0.34:0.02:0.5 ]                    (44)

a = [ ones(1,15), zeros(1,15) ]                         (45)

[he,ho] = amp(f,a)                                      (46)
```

Here the first 15 samples have amplitude 1 and are spaced at 0.016 in frequency. This is followed by six samples of amplitude 0, these six again spaced at 0.16. Finally, there are nine more samples of amplitude 0, these spaced at 0.02 in frequency. It might seem that changing the spacing from 0.16 to 0.20 would be a relatively minor thing. In fact, we see the closer spacing in the passband resulting in a relatively smooth response, and this extends a bit into the stopband. However, this is followed by an extreme blow-up in the stopband centered about 0.4 as the spacing increases from 0.016 to 0.02. Since we expect to improve some band by using closer spacing, we need to be aware of the associated decrease in performance in other bands that have wider spacing.

7. USING MORE EQUATIONS THAN UNKNOWNS

In some cases, it is useful to consider a number of frequency sampling points that is greater than the length of the filter to be designed. For example, if the filter is of length N, we might still want to take M sample points where M is greater than N. In such a case, we have more equations that unknowns, and we can not solve these in the usual way (matrix inversion) but must instead use the least square procedure or pseudo-inverse. In the case where M=N, we can fit the response exactly to the M points for zero error at each point and zero error total. For the case of M>N, we expect the curve in general to not go exactly through any of the specified points. In this case of M>N, in general, there is an error at each point, and we seek to minimize this error in the least squares sense.

This least square procedure is well presented [2] in terms of the use of a "pseudo-inverse" and has been applied to the frequency sampling filter design problem [3]. The LMS program, fsamplms.m is quite similar to the other programs presented above. The phase is set in a similar way, but it is important to recognize that while the frequency samples divide 2π by M, it is still N, the length of the filter we end up with, that determines the linear phase term. The matrix corresponding to the coefficients of the equations is set up in a manner similar to gfs.m, the program for unequal spacing. Here the samples are equally spaced, but we must still use the matrix (instead of the DFT) because in general $N \neq M$. The major difference is thus that the pseudo-inverse, $(E^tE)^{-1}E^t$, is used instead of just E^{-1}.

PROGRAM 4: fsamplms.m

```
function h = fsamplms(N,M,f,a,m)
%    function h = fsamplms(N,M,f,a,m)
%                 (MATLAB filter input format)
%    N = length
%    M = number of freq. samples
%    f = frequency vector on 0-1 with 1 = half the sampling freq.
%    a = amplitude vector
%
%    B. Hutchins                                        Fall 1995
%

i=0:M-1;
wi=2*pi*i/M;
%------------------generate phase-----------------
ph=exp(-j*wi*(N-1)/2);
if mod(N,2)==0;
  ps=[ones(1,M/2) -ones(1,M/2)];
  ph=ph.*ps;
end

%-------------generate default amplitude------------
for n=1:M
       if wi(n) < f(2)*pi; mask(n)=a(2);
          elseif wi(n) > (2*pi-f(2)*pi); mask(n)=a(2);
          else mask(n)=a(3);
       end
end
if exist('m')==1 mask=m; end;
H=mask.*ph;

%--------------------------------------------------
k=0:N-1;
arg=-j*(wi'*k);
E=exp(arg);

h=(inv(E'*E))*E'*H.';

figure(1);stem(k,h)
MH=abs(freqz(h,1,500));
MH=MH/MH(1);
figure(2);plot([0:.001:.499],MH)
MHDB=db(MH,120,20);
MHDB=MHDB-MHDB(1);
figure(3);plot([0:0.001:.499],MHDB)
grid
figure(3)
```

There are several circumstances where we may wish to use an over-determined set of samples. In the first instance (Example 14), we simply wish to better define a band or bands by using more samples. In a second instance (Example 15), we may wish to determine the general order of an FIR filter that has a magnitude response similar to a particular IIR filter. For example, by taking amplitude samples from an IIR Butterworth response, we can have the same well-defined magnitude response with an imposed linear phase replacing the Butterworth's normal phase. These we will look at one at a time.

In Example 14a (Fig. 17a), we use the command line:

h = fsamplms(19,19, 2*[0 .25 .25 .5],[1 1 0 0]) (47)

which reverts to an equal spaced case with 19 equations in 19 unknowns. The result is the same as Example 1, and is used here for reference and to remind us that the DFT now becomes a special case of the LMS and pseudo-inverse procedure. As noted, the rejection at the first sidelobe is about -16db. The contrasting case is Example 14b (Fig. 17b) where we have used:

h = fsamplms(200, 19, 2*[0 .25 .25 .5],[1 1 0 0]) (48)

which gives us 200 samples reducing to a length 19 filter. The result is a response with something like -21db rejection of the first sidelobe. Further, and possibly more importantly, the cutoff region is now somewhat closer to the specified 0.25 because samples are taken on a finer grid.

Fig. 18a, 18b, and 18c constitute Example 15. In total, this procedure amounts to the imposition of a linear phase on an IIR Butterworth magnitude. It also can serve to compare the order of an FIR filter that is required to match the roll-off rate of an IIR filter. In this procedure, we have started with a 12th order analog Butterworth low-pass and converted this prototype to an IIR digital low-pass with cutoff at $f_s/4$, using Bilinear z-Transform. The magnitude response of this IIR filter is then calculated at 500 points and stored to become the input samples for the frequency sampling filter (using the 'whole' option with freqz to generate both halves of the response).

For Fig. 18a, we have used:

h = fsamplms(12,500,[0 0.5 .5 1],[1 1 0 0],HB) (49)

where HB is the vector of 500 samples of the Butterworth magnitude. [Note that numbers specified for the frequency and amplitude vectors, in the [ ] of equation (49), are irrelevant since they are overridden by the inclusion of the HB parameter.] Here while we use all 500 samples, we reduce the final result to a length 12 filter. Thus we have a reasonable comparison of a 12th order IIR vs. a 12th order FIR, with the FIR clearly losing on the magnitude specifications.

Figures 18b and 18c are similar in that the same 500 samples are input, but the filter lengths are now 25 and 50, respectively. With length 25, the FIR filter looks a good deal better than the length 12 case. The length 50 FIR filter is getting very close to the IIR Butterworth. Accordingly, we might suppose that the FIR filter needs to be something like an order of magnitude longer than the corresponding IIR to get a roughly equivalent magnitude response. However, here we have used the cutoff at $0.25f_s$, and this is the easiest case to match. The procedure for additional study is probably clear.

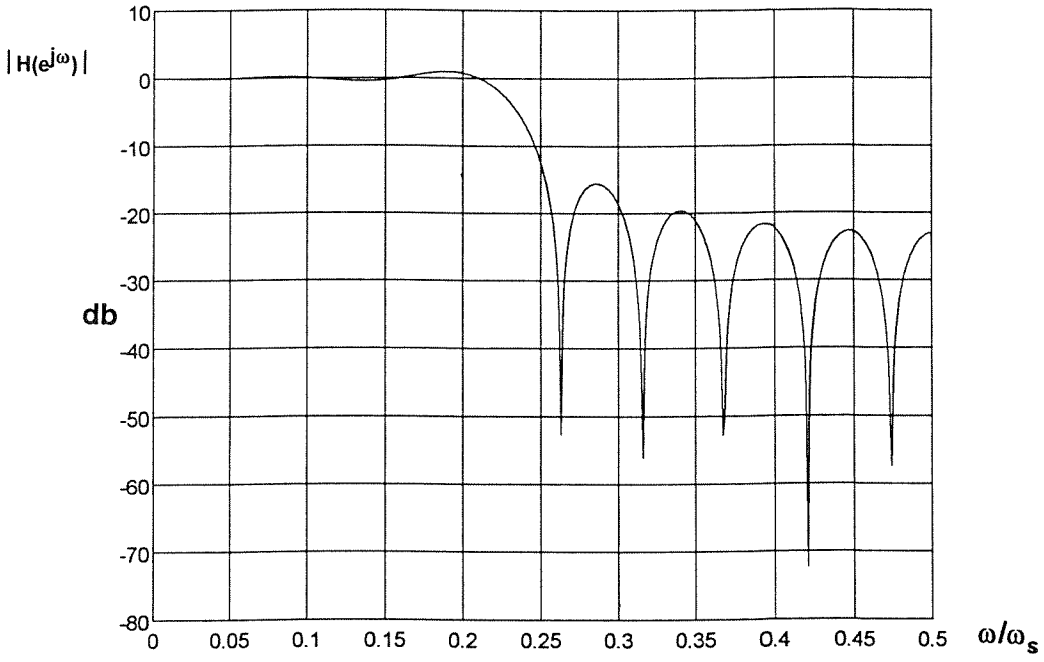


Fig. 17a Example 14a: 19 tap filter from 19 samples (same as Example 1)

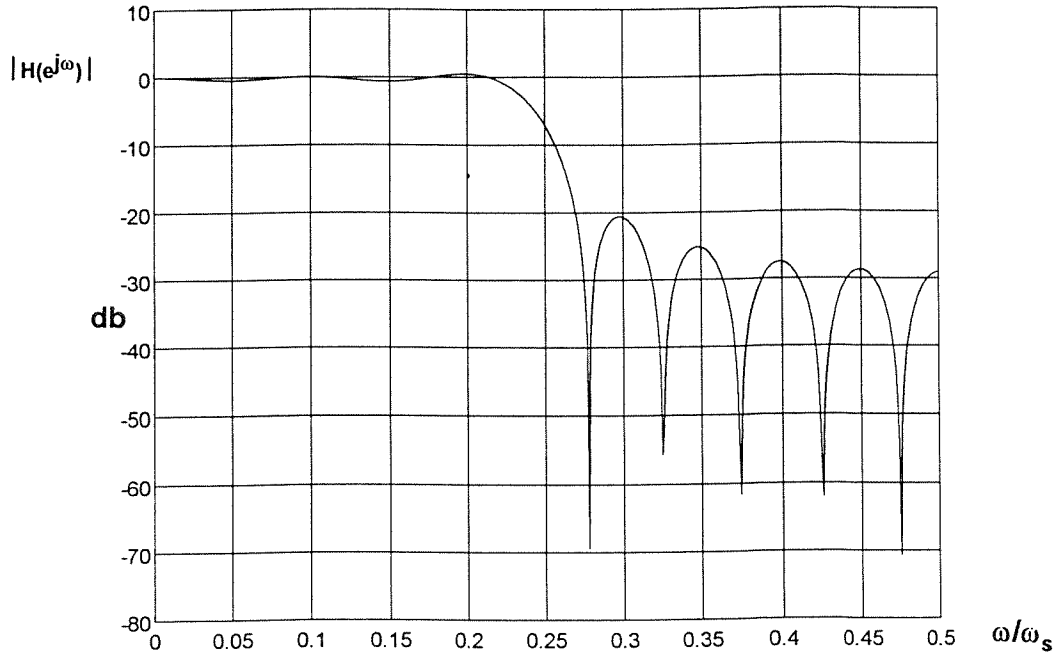


Fig. 17b Example 14b: 19 tap filter from 200 samples

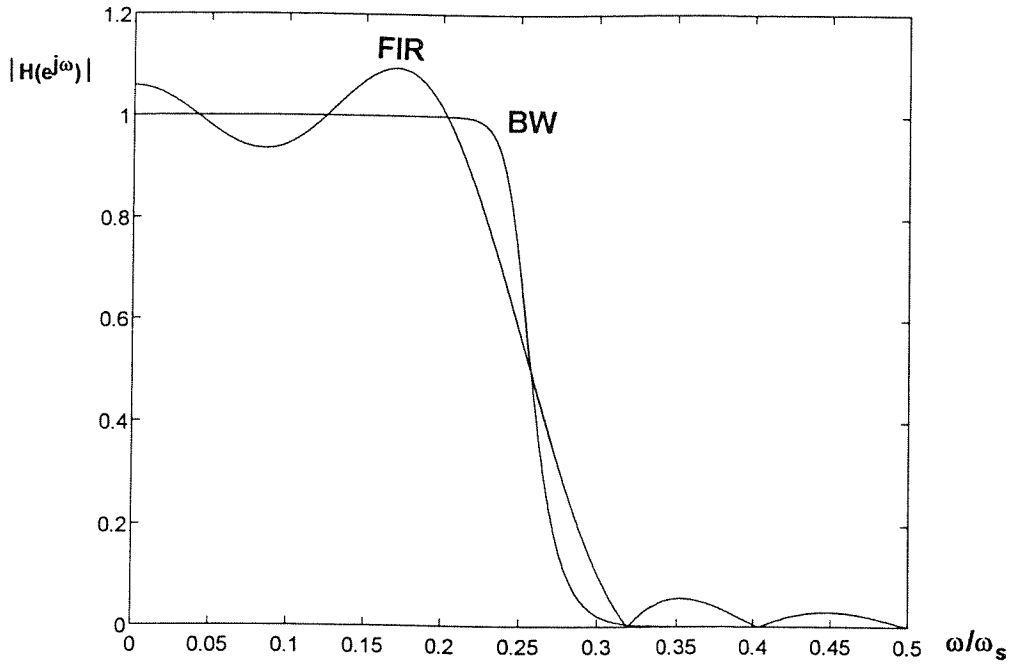


Fig. 18a Example 15a: Length 12 FIR filter trying to be order 12 IIR Butterworth

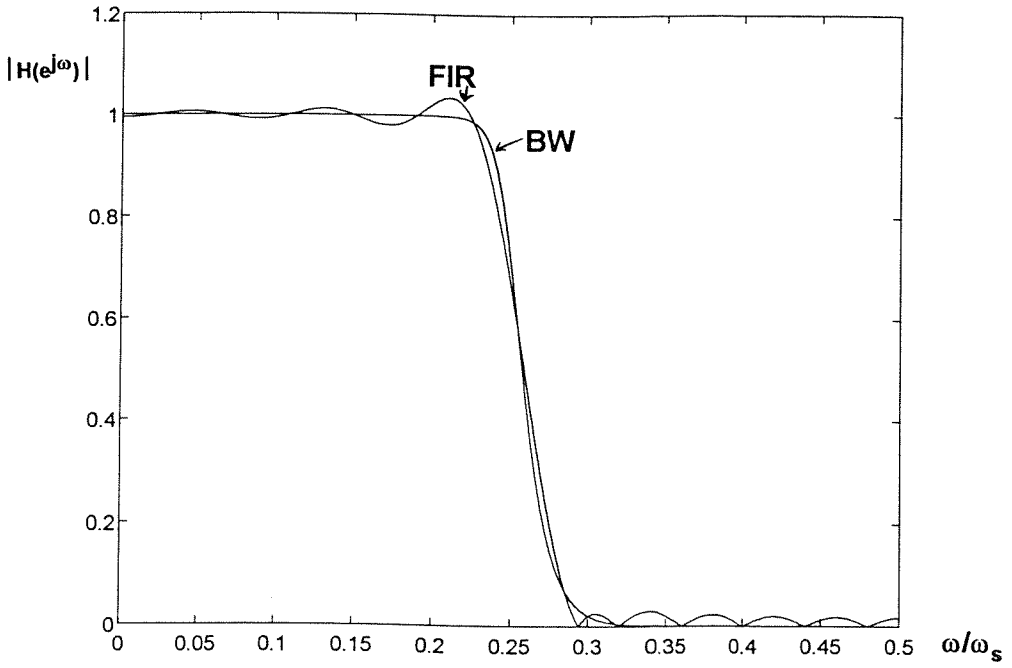


Fig. 18b Example 15b: Length 25 FIR filter trying to be order 12 IIR Butterworth

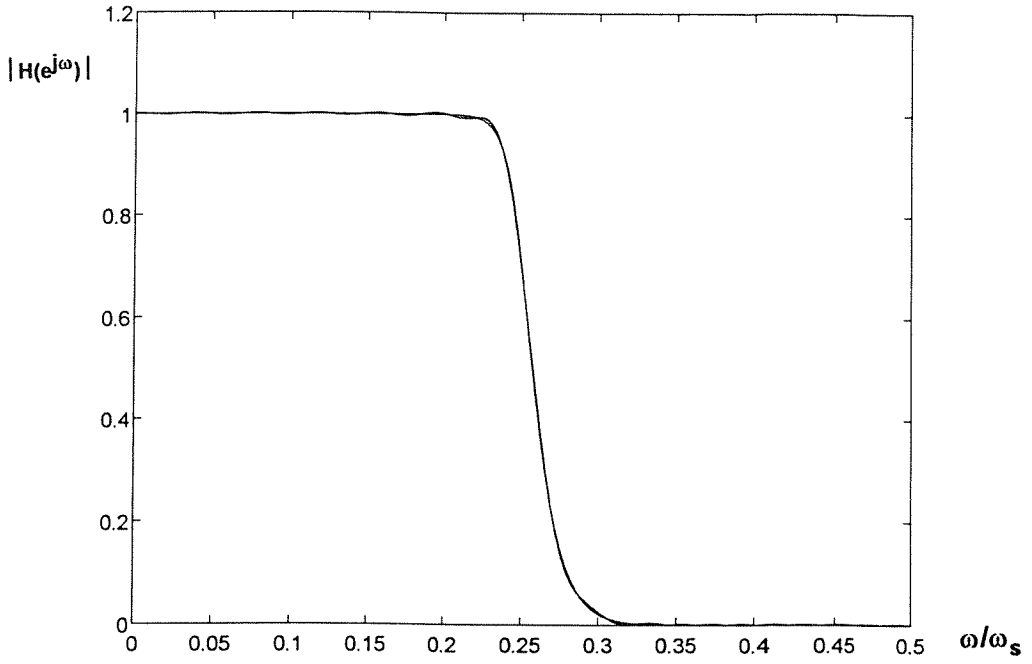


Fig. 18c Example 15c: Length 50 FIR filter trying to be order 12 IIR Butterworth

8. WEIGHTED LEAST SQUARES

The method of least squares in Section 7 above can be modified to accept a weight on the error in different bands, or even on individual samples if we wish. We simply have to determine a weight vector W much as we do an amplitude vector for each sample. Next, representing W as a diagonal matrix, we can find h by using $(E^tW\,E)^{-1}E^tW$, replacing $(E^tE)^{-1}E^t$ for the unweighted case, replacing E^{-1} for the case where M=N. An example program, fsampwls.m is shown as Program 5. Perhaps to our surprise, the actual weighting only works when M>N and only well when M is something like (at least) twice N. For M=N, the weighting has no effect. For example, a weight vector of 1000 on the passband and 1 on the stopband will give exactly the same thing as 1 on the passband and 1 on the stopband (etc.). No flattening of the passband occurs as a result of the larger weighting. The reason for this is that when M=N, there is an exact, zero-error solution (the designed response goes exactly through the desired response). In consequence, no weighting of this zero error makes any difference.

Choosing M>>N, we can use the weighting vector much as we did the weighting with integrated least squared error design [4]. If M=N and we need to

flatten a band, unequal spacing of samples rather than weighting is suggested. No examples of this method or program are given here.

PROGRAM 5: fsampwls.m

```
function h = fsampwls(N,M,f,a,w)
%    function h = fsampwls(N,M,f,a,w)
%
%         Frequency Sampling with Weighted Least Squares
%
%                 (MATLAB filter input format)
%    N = length
%    M = number of freq. samples
%    f = frequency vector on 0-1 with 1 = half the sampling freq.
%    a = amplitude vector
%    w = weights
%
%    [ Note: M must be greater than N for the weighting to
%      work.  If M=N then the error is minimized to exactly
%      zero for any choice of weighting. ]
%
%    B. Hutchins                                    Fall 1995
%

i=0:M-1;
wi=2*pi*i/M;
%------------------generate phase-----------------
ph=exp(-j*wi*(N-1)/2)
if mod(N,2)==0;
  ps=[ones(1,M/2) -ones(1,M/2)];
  ph=ph.*ps;
end

%----generate default amplitude and weight vectors--------
for n=1:M
       if wi(n) < f(2)*pi; mask(n)=a(2);ww(n)=w(1);
          elseif wi(n) > (2*pi-f(2)*pi); mask(n)=a(2);ww(n)=w(1);
          else mask(n)=a(3);ww(n)=w(2);
       end
end
mask;
H=mask.*ph;
ww=diag(ww);

%-------------------------------------------------
k=0:N-1;
arg=-j*(wi'*k);
E=exp(arg);

h=(inv(E'*ww*E))*(E'*ww*H.')

figure(1);stem(k,h)
MH=abs(freqz(h,1,500));
MH=MH/MH(1);
figure(2);plot([0:.001:.499],MH)
MHDB=db(MH,120,20);
MHDB=MHDB-MHDB(1);
figure(3);plot([0:0.001:.499],MHDB)
grid
figure(3)
```

9. A FINAL EXAMPLE

We close with a final example. Here we will use gfs.m, unequal spacing, and with three bands as:

```
f=[0 .05 .1 .15 .2 .25 .315 .35 .4 .425 .45 .5 .55 .575 .6 .65 .685 .75 .8 .85 .9 .95]   (50)

a=[1 1 1 1 1 0 0 0 .5 .5 .5 0 .5 .5 .5 0 0 0 1 1 1 1]                                    (51)

h=gfs(f,a)                                                                               (52)
```

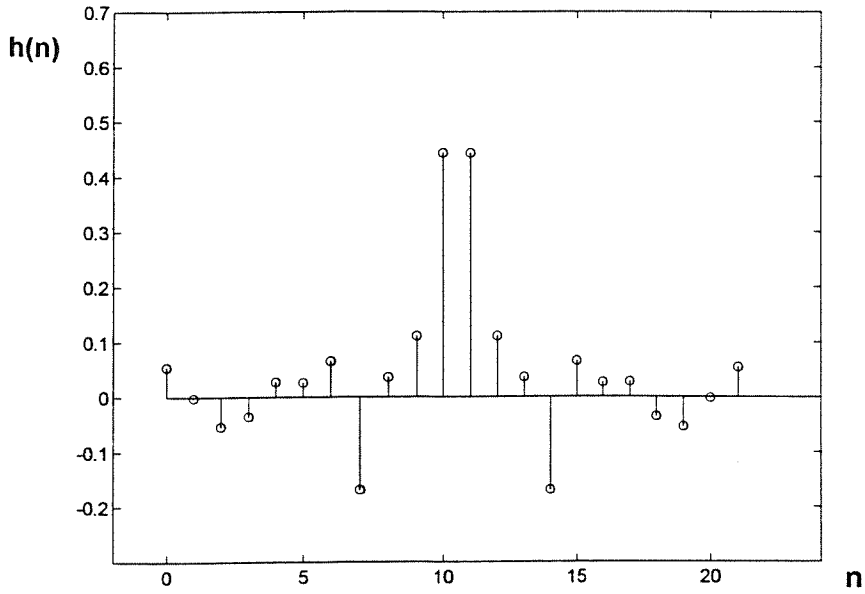


Fig. 19a Impulse response for Example 16

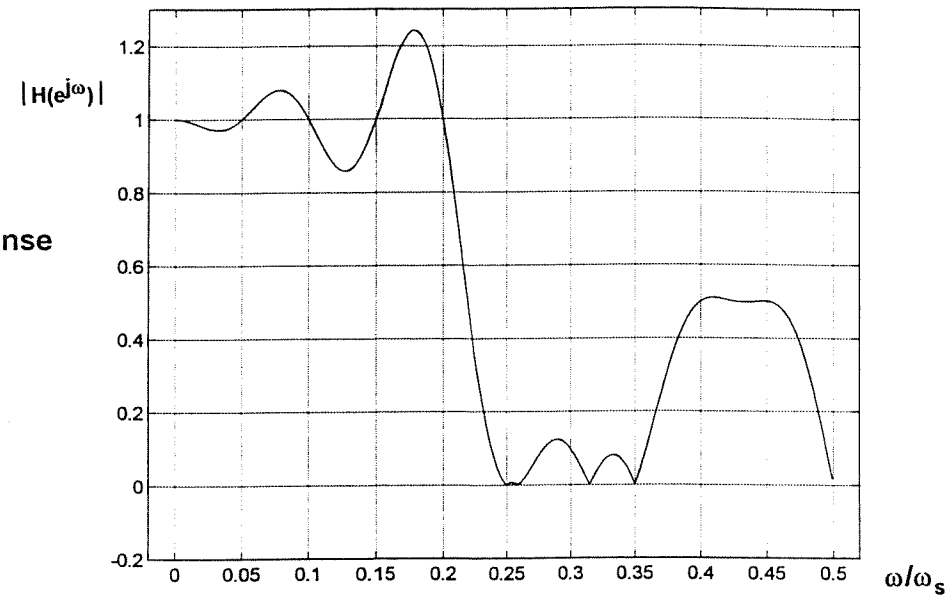


Fig. 19b Magnitude response for Example 16

Here we have placed a sample of value zero (that might be expected, with equal spacing, to be at 0.3) at 0.315, supposedly to null out a component exactly at that frequency. In addition to the passband from about 0 to 0.2, and a stopband from about 0.25 to 0.35, we have added a third band of amplitude 0.5 just above 0.4. Note however that because we have an even length filter, we need to bring this band down to 0 at the frequency 0.5. Further here, a tighter spacing of frequencies in this third band results in a flatter response there as compared to the first two bands.

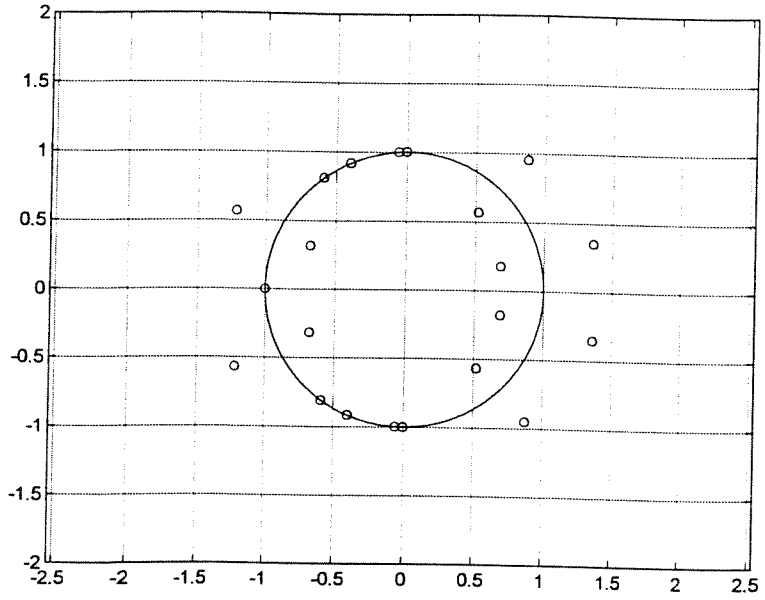


Fig. 19c Zeros corresponding to Example 16

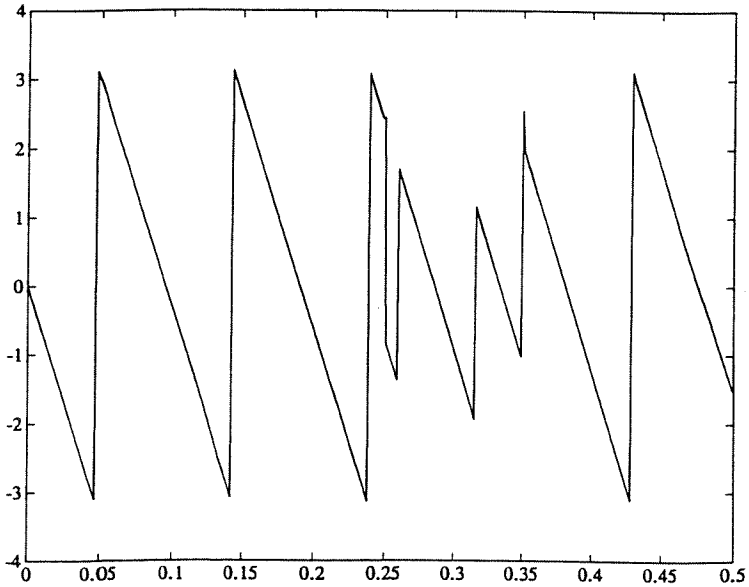


Fig. 19d "Angle" of phase for Example 16

Example 16 is illustrated by Fig. 19a, the impulse response; Fig. 19b, the magnitude of the frequency response; and by Fig. 19c, the plot of the zeros of the response. Also shown are the angle (Fig. 19d), the phase (Fig. 19e), and the group delay (Fig. 19f). The impulse response is clearly linear phase, even symmetry as expected. The magnitude response shows the expected (specified) features to the accuracy we expect for the length chosen. The zero plot shows zeros (pairs) corresponding to the specified zero samples (at 0.25, 0.315, and 0.35) but we also get an additional zero (pair) just above 0.25, and the "automatic" zero at 0.5 due to the even length.

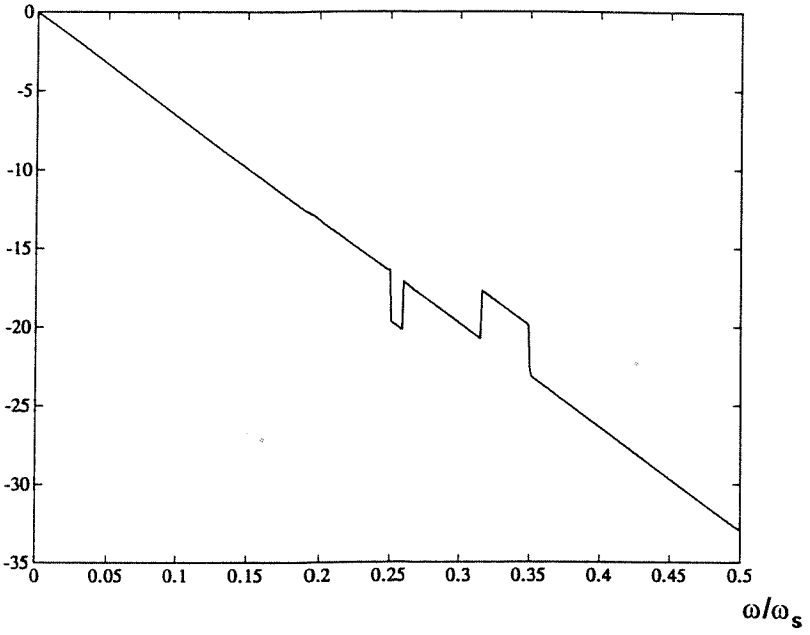


Fig. 19e "Phase" plot for Example 16

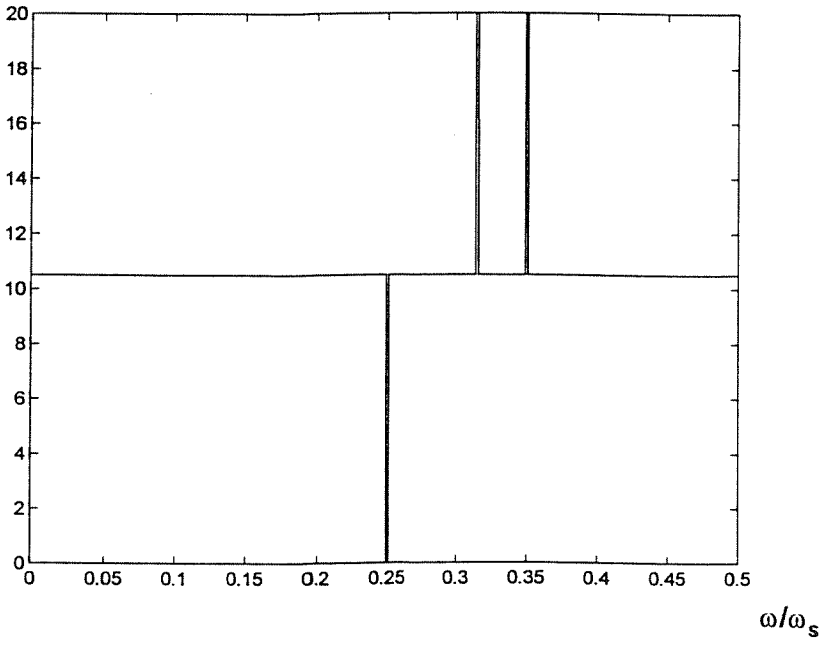


Fig. 19f Group delay of Example 16

When we look at the phase, we generally have several options. The MATLAB "angle" function is one of these, which returns the arctan of the imaginary part divided by the real part of the transfer function. This can be somewhat cluttered due to resets every 2π. The MATLAB "phase" function tries to (and does) improve things by trying to "unwrap" the 2π phase jumps. However, as always, the phase response can be ambiguous.

Less ambiguous and more to the point is the plot of group delay, the negative derivative of the phase with respect to frequency. Both the angle and the phase show us that the group delay is flat (the slope is constant) except at certain points where it is infinite (the zeros on the unit circle). Note also that the group delay is exactly (N-1)/2=10.5, the specified delay of the linear phase.

REFERENCES

[1] Anon, The Student Edition of MATLAB, Prentice-Hall (1992)

[2] L. B. Jackson, Digital Filters and Signal Processing, (Second Edition) Kluwer Academic Publishers (1989) pp 249-252

[3] T.W. Parks & C.S. Burrus, Digital Filter Design, Wiley (1987), pp 44-48

[4] B. Hutchins, "Weighted, Integrated, Least-Squared Error for FIR Filter Design, " Electronotes Application Note AN-332, August 1995

The following two references are excellent summaries of frequency sampling:

[5] L.R. Rabiner & B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall (1975), pp 105-123

[6] C.S. Burrus et al, Computer Based Exercises for Signal Processing, Prentice-Hall, (1994), pp 249-261