ELECTRONOTES
1016 Hanshaw Rd
Ithaca, NY 14850
607-266-8492

APPLICATION NOTE NO. 366

May 2006

UNFILTERING – EQUALIZATION

1. INTRODUCTION

A filter modifies a signal – that is its purpose. But if it is a linear time-invariant (LTI) system, it only modifies the amplitude and phases of input components, but does not change any frequencies. These modifications are according to what we call its "frequency response." Yet often times this view is not particularly helpful. Such is the case where we have relatively short transient-like inputs for which the Fourier decomposition is continuous in frequency and possibly hard to determine. In such cases, a time-domain view may work better.

For example, we might have a sharp rectangular pulse (or sum of such pulses) as the input. Moreover, the filter might be unintentional. We might be trying to transmit the pulse over a "channel" and the channel is not "transparent." The sharp pulse gets smeared, and may not retain enough information about its presence and location at the far end of the channel. Such a channel can usually be characterized as a system or filter. Our interest in such cases is often one of undoing the damage done by the channel, something that is often called "equalization."

2. INVERSE FILTER

Clearly one simple approach to this is to consider using the "inverse filter" if the original filter is known (perhaps by analysis of by experiment). For example, suppose that the first filter is a length-3 FIR with tap weights 1, 2/3, and 1/3, perhaps the sort of thing that we might think of as "smearing." That is, if we put in an impulse, the response to the input is three output values: 1, 2/3, and 1/3. Thus

$$H_1(z) = 1 + (2/3)z^{-1} + (1/3)z^{-2} \qquad (1)$$

We can understand that this could cause problems. If for example, we wanted to clean up the signal on the receiving end, we might threshold the samples relative to 1/2, and what should have been a single impulse becomes two adjacent impulses.

Thus if we know the original filter, we can easily calculate its inverse, which is here a purely IIR filter, as:

$$H_2(z) = 1 / [1 + (2/3)z^{-1} + (1/3)z^{-2}] \qquad (2)$$

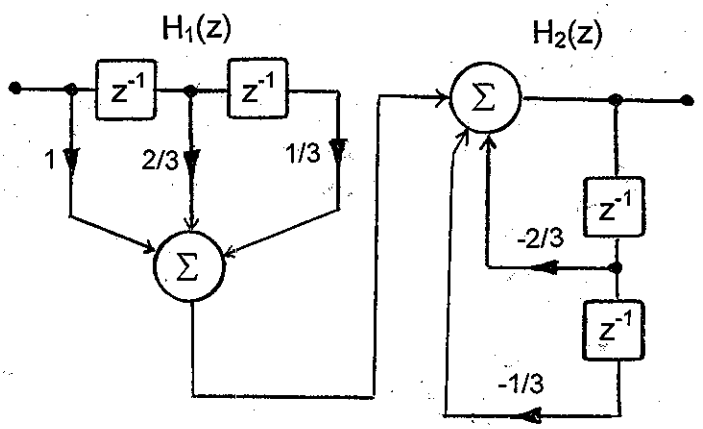

Fig. 1a "Channel" $H_1(z)$ and Equalizer $H_2(z)$

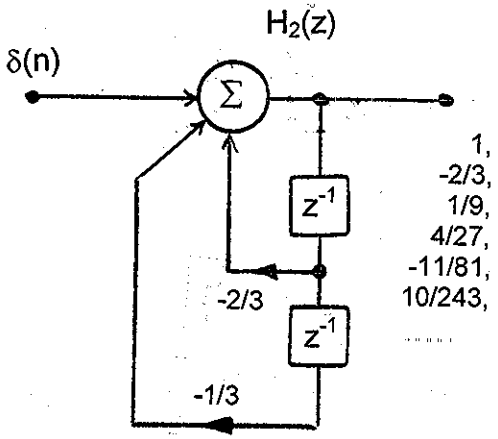

Fig. 1b Equalizer $H_2(z)$

It is obvious that the product $H_1(z)H_2(z) = 1$ and is transparent. Fig. 1a shows the case where $H_1(z)$ is in series with $H_2(z)$ and if we look at the impulse response of this cascade, we see that it is just the impulse itself – sort of magic the way the IIR section is shut down (this is easily done by simulation by hand). Note from Fig. 1b that the IIR section by itself, does have an infinite duration impulse response.

n	$h_2(n)$
0	1.0000
1	-0.6667
2	0.1111
3	0.1481
4	-0.1358
5	0.0412
6	0.0178
7	-0.0256
8	0.0111
9	0.0011
10	-0.0045
11	0.0026
12	-0.0002
13	-0.0007
14	0.0005

etc.

This inverse filter "answer" serves as a guide for how well other filters may serve as an equalizer for $H_1(z)$.

$H_2(z)$ in this case is a stable IIR filter, which is seen from the decaying impulse response, or from the poles which are at [same as the zeros of $H_1(z)$ of course]:

$$p_{1,2} = -0.3333 \pm 0.4714\,j \tag{3}$$

which are nicely inside the unit circle.

This inverse filter works even when the inverse filter has poles on, or even outside the unit circle – at least in theory. This is because the unstable poles come after the corresponding zeros that cancel them. In practice, this is not good enough, because of noise in the output of the channel. Let's look at a second related example. Consider

$$H_1(z) = 1/3 + (2/3)z^{-1} + z^{-2} \tag{4}$$

with it's corresponding inverse

$$H_2(z) = 1 / [1/3 + (2/3)z^{-1} + z^{-2}] \tag{5}$$

so that now the poles of $H_2(z)$ [zeros of $H_1(z)$] are at:

$$p_{1,2} = -1.0000 \pm 1.4142\,j \tag{6}$$

which are the reciprocals of the poles in the first example, and clearly outside the unit circle. We can modify the setup of Fig. 1a as shown in Fig. 2, which shows the new filters and the addition of a noise to the output of the first filter. Simulation of this with noise amplitude zero shows recovery of the impulse (Fig. 3a) while even a tiny bit of noise (amplitude 0.0000002 relative to a unit impulse) causes the output to blow up (Fig. 3b). What this says essentially is that if you take an unstable filter [$H_2(z)$], with its internal states initially zero, and give it any input that has not been pre-filtered to cancel [by $H_1(z)$], the filter $H_2(z)$ will blow up. The noise goes directly into $H_2(z)$, and thus gets the filter started blowing up.

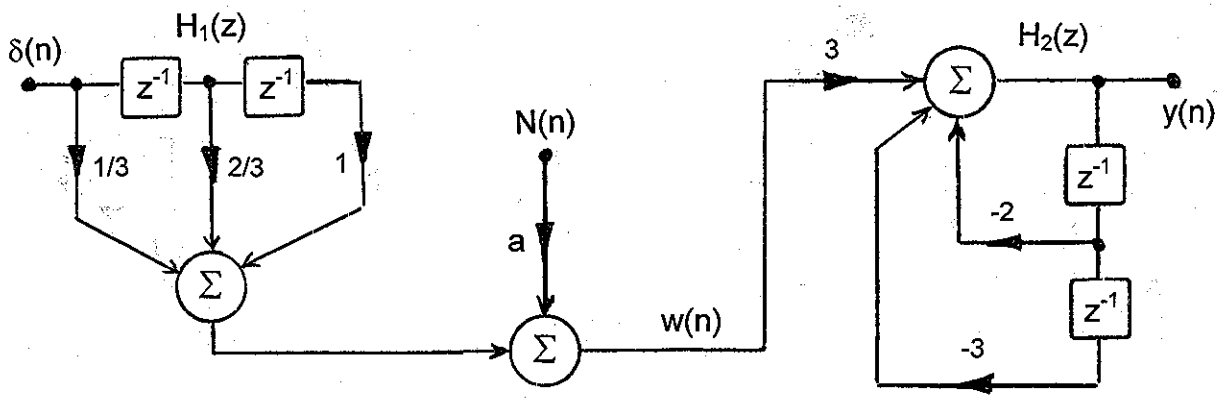

Fig. 2 "Channel" $H_1(z)$ and unstable Equalizer $H_2(z)$ with possible noise N(n)

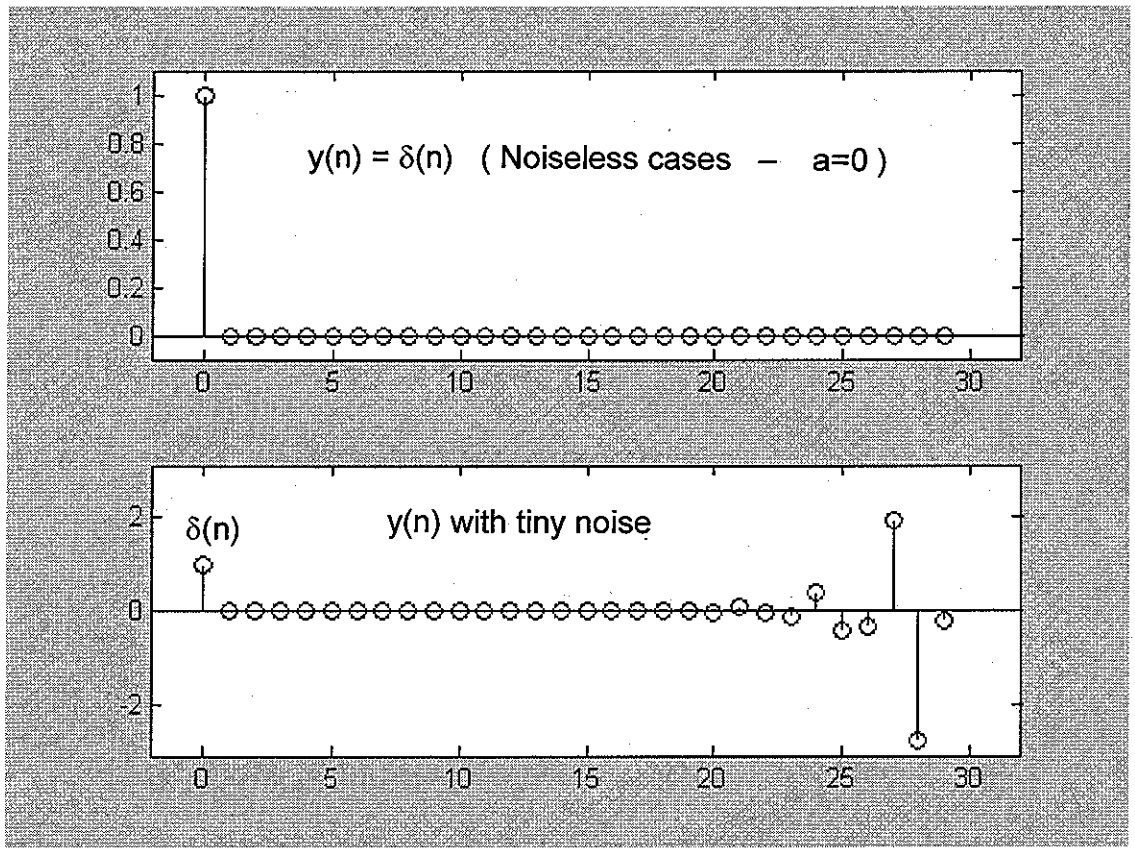

Fig. 3 In (a), the unstable inverse $H_2(z)$ works because the impulse was pre-filtered by $H_1(z)$, and the noise is zeroed. In (b), the noise is non-zero (still tiny) but it is enough to cause the filter $H_2(z)$ to start blowing up.

One way to obtain a FIR approximation to an IIR response is to truncate the IIR impulse response to finite length, but this clearly would work only for the case where the response is getting small with increasing time, and where enough impulse response values are kept so that the truncation error is minimized. If the IIR filter is unstable (for positive time) this is never going to work.

3. LEAST SQUARES FIR EQUALIZATION

We could take the approach of insisting that the equalizer $H_2(z)$ be FIR. Our goal would be that the impulse response of the cascade of $H_1(z)$ with $H_2(z)$ would be, or more likely would approximate, a delayed (and possibly scaled) impulse. If we input an impulse $\delta(n)$ into the cascade, the output is the convolution of $h_1(n)$ with $h_2(n)$. We would like this convolution to consist of values that are all zero except for one value which

is 1. We know $h_1(n)$ and we choose a desired length for $h_2(n)$. The length of the convolution of $h_1(n)$ with $h_2(n)$ [which we can all $h_3(n)$] is the sum of the lengths of $h_1(n)$ and $h_2(n)$ minus 1. Thus since the length of $h_1(n)$ is two or more (otherwise, there is no problem to solve), we see that we need to choose values of a convolution sequence that is longer than $h_2(n)$. We do not have enough information for an exact solution.

Consider the following example. Suppose $h_1(n)$ is the sequence {1 1}. That is, the filter smears an input sequence by giving an output that is the sum of two consecutive samples. Thus we have $H_1(z) = 1+z^{-1}$. The inverse filter to this would be $H_2(z) = 1/(1+z^{-1})$ so it would have a pole at z=-1 (on the unit circle) and would not be an acceptable inverse.

So we choose a possible FIR as an equalizer. Suppose we choose a length four FIR for $H_2(z)$ with tap weights a, b, c, and d (to be determined). The convolution of $h_1(n)$ and $h_2(n)$ is thus length 5, and we might choose this to be a target sequence $h_3(n)$ = {0 0 1 0 0} (a twice delayed impulse). How do we choose a, b, c, and d?

The first value of the convolution of a sequence {1 1} with a sequence {a b c d} is just a and we want this to be the target value of 0. Thus a=0. The second term of the convolution sequence is a+b and we also want this to be 0. The third term of the convolution is b+c and we want this to be 1. The fourth term is c+d which should be 0, and the fifth term is d which should be 0. This is not going to work! In summary, we have 5 equations, but only 4 unknowns.

$$\begin{aligned} a &= 0 \\ a+b &= 0 \\ b+c &= 1 \\ c+d &= 0 \\ d &= 0 \end{aligned} \tag{7}$$

which are, in matrix form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

All we can do is solve these to minimize the squared error. Note that the errors are:

(a-0)
(a+b-0)
(b+c-1)
(c+d-0)
(d-0) (9)

If we square these and sum them, the total squared error which we can call E^2 is:

$$E^2 = 2a^2 + 2b^2 + 2c^2 + 2d^2 - 2b - 2c + 2ab + 2bc + 2cd + 1 \quad (10)$$

We can take partial derivatives of E^2 with respect to a, b, c, and d, and set them equal to 0. This gives us:

$$\partial E^2/\partial a = 4a + 2b = 0$$

$$\partial E^2/\partial b = 4b - 2 + 2a + 2c = 0 \quad (11)$$

$$\partial E^2/\partial c = 4c - 2 + 2b + 2d = 0$$

$$\partial E^2/\partial d = 4d + 2c = 0$$

We now have four equations in four unknowns which are, in matrix form

$$\begin{bmatrix} 4 & 2 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 0 & 2 & 4 & 2 \\ 0 & 0 & 2 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 2 \\ 2 \\ 0 \end{bmatrix} \quad (12)$$

which invert to

$$h_2 = \{ a\ b\ c\ d \} = \{ -0.2 \quad 0.4 \quad 0.4 \quad -0.2 \} \quad (13)$$

In order to see if this is helping, we need to see how well we approach our target convolution sequence of { 0 0 1 0 0}. Convolving {-0.2 0.4 0.4 -0.2} with the h_1 sequence {1 1} we arrive at:

$$h_3 = \{ -0.2 \quad 0.2 \quad 0.8 \quad 0.2 \quad -0.2 \} \quad (14)$$

This is not all that bad for a length 4 equalizer. Note for example that if we were to threshold this output relative to 0.5, we would have our targeted result.

4. A GENERAL PROGRAM

Above we solved the equalization problem with a classic least squares approach, and we set up the problem and solved it, largely by hand. It is useful to do this, but we might well get tired of doing many examples, and we do need more examples to see how well this can work. We will short-cut by using a couple of Matlab functions.

In our hand example, we wrote out the elements of the convolution sum, equation (8), and then wrote these as a matrix, equation (8), which is a convolution matrix. It is clear how we do this, and the matrix elements are easily written down for specific cases (and are certainly not limited to being all ones or having any particular pattern). Matlab has a function ***convmtx*** that does this. The least squares solution is also easily solved with the Matlab pseudo-inverse, ***pinv***. It is then merely a matter of writing the target convolution, which differs slightly for even and odd cases. In fact, this target could have been made an input parameter to the program, and need not be made just a delayed impulse. These possible changes to the program are likely obvious. The program is given below as lsi.m.

```
function [h1,h2,h3]=lsi(h1,N)
% function lsi(h1,N)
%     h1 = FIR to invert
%     N  = length of inverting FIR
%     h2 = inverting FIR (to calculate)
%
L=length(h1)+N-1    % convolution length h1 and h2

m=convmtx(h1,N)'

sm=size(m);
r=zeros(1,sm(1));
lr=length(r);

if mod(lr,2)==1;
    r((lr+1)/2)=1
end
if mod(lr,2)==0;
    r(lr/2)=1
end

% least squares inversion
h2=pinv(m)*r';
% test - does this look like r
h3=conv(h2,h1);
h2=h2';
h3=h3';
```

5. MORE EXAMPLES

Using the general program, we can run through a number of examples of least square FIR inversion. We can begin by repeating the hand example of Section 3, but do it for a longer length equalizer. Here the command lines is: lsi([1 1],14), so our equalizer is length 14. The result is seen in Fig. 4

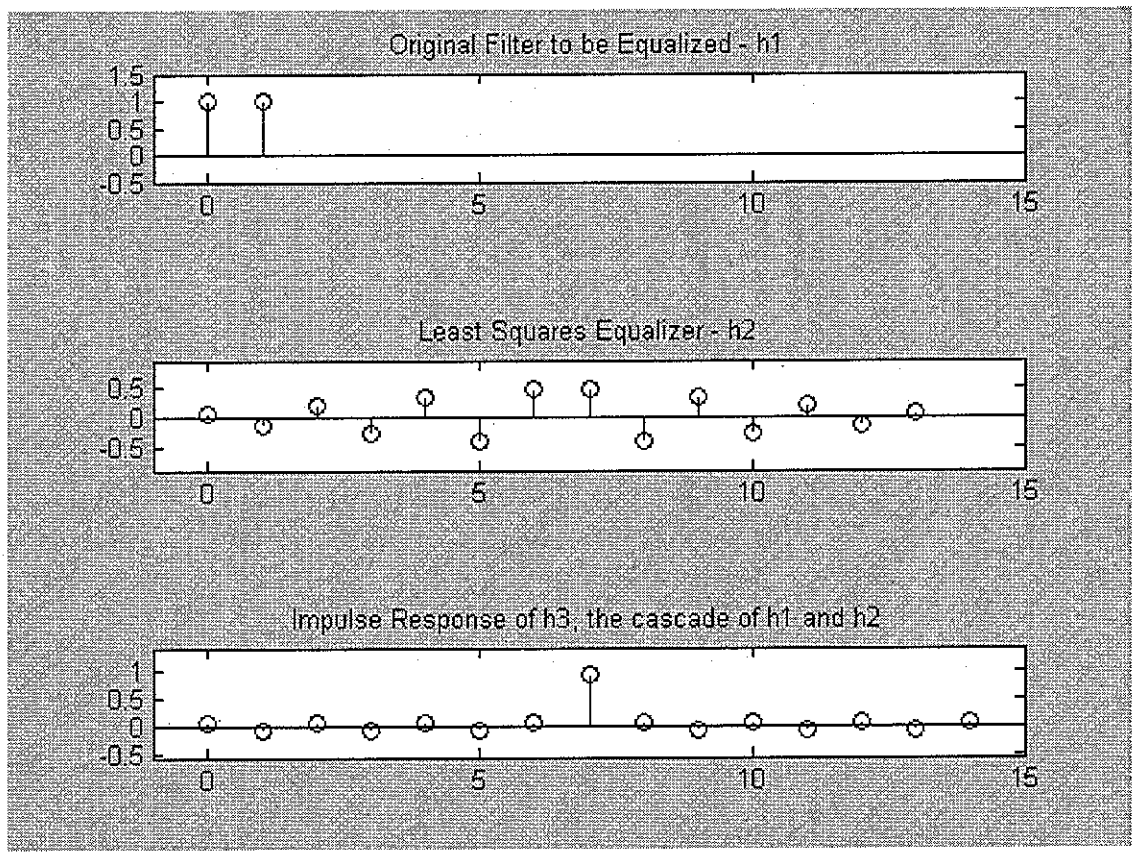

Fig. 4 lsi([1 1],14) Length 14 equalizer of length 2 hold.

In our length-4 hand-calculated case, we had

$$h_2 = \{ -0.2 \quad 0.4 \quad 0.4 \quad -0.2 \} \tag{15}$$

and

$$h_3 = \{ -0.2 \quad 0.2 \quad 0.8 \quad 0.2 \quad -0.2 \} \tag{16}$$

In this case we have:

$$h_2 = \{0.0667 \quad -0.1333 \quad 0.2000 \quad -0.2667 \quad 0.3333$$

$$-0.4000 \quad 0.4667 \quad 0.4667 \quad -0.4000$$

$$0.3333 \quad -0.2667 \quad 0.2000 \quad -0.1333 \quad 0.0667\} \qquad (17)$$

and

$$h_3 = \{\ 0.0667 \quad -0.0667 \quad 0.0667 \quad -0.0667 \quad 0.0667$$

$$-0.0667 \quad 0.0667 \quad 0.9333 \quad 0.0667 \quad -0.0667$$

$$0.0667 \quad -0.0667 \quad 0.0667 \quad -0.0667 \quad 0.0667\} \qquad (18)$$

We see that the longer length equalization filter results in an overall impulse response (h_3) that is more like a delayed impulse.

As a second example of the use of the program, consider lsi([1 2/3 1/3],15) which has the same $h_1(n)$ as was used in Section 2, and we saw that this had a stable inverse. Fig. 5 shows the result, which appears in the figure, to be excellent. Looking at the actual numbers, we find that for the equalizer we get:

$$h_2 = \{\ 0.0000 \quad 0.0000 \quad -0.0000 \quad 0.0000 \quad 0.0000 \quad -0.0000 \quad 0.0000 \quad -0.0000$$

$$0.9999 \quad -0.6664 \quad 0.1109 \quad 0.1476 \quad -0.1342 \quad 0.0395 \quad 0.0163 \quad -0.0178\} \qquad (19)$$

which is a good approximation (delayed) to the true inverse filter response tabulated in Section 2. This is expected, and the approximation gets better and better as we increase the length of the FIR equalizer. The response $h_2(n)$ is properly thought of as truncated FIR approximation to the actual IIR inverse.

Naturally our interest now turns to the case where there was an unstable inverse. What happens if we try: lsi([1/3 2/3 1], 15) ? The result is shown in Fig. 6 and we note that $h_3(n)$ in this case is indeed a delayed impulse, so this does "work." It is interesting to see how it worked, and this we learn from $h_2(n)$ of Fig. 6. <u>It's the "anti-causal" part of the blown up response.</u> A filter that blows up for positive time decays for negative time (and vice versa). We have just turned things around. Reversing $h_1(n)$ forced us to reverse $h_2(n)$. Note that just as it was necessary to not truncate the FIR response until it got small on the positive side, here we need to be sure we start the FIR equalizer where the response is small (small relative to input signal levels) on the negative side.

In fact, this tells us how to handle the issue of an unstable inverse – by taking the anti-causal portion, truncating it at a negative value of n where it is sufficiently small, and shifting the whole response to the right to make it causal.

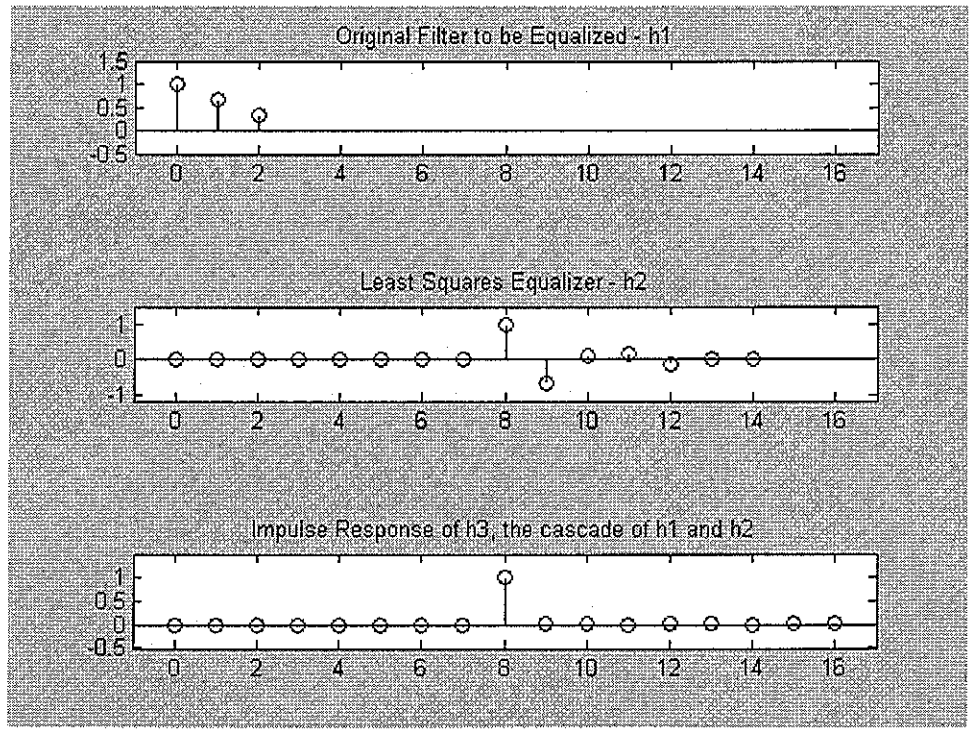

Fig. 5 lsi([1 2/3 1/3], 15) - stable inverse

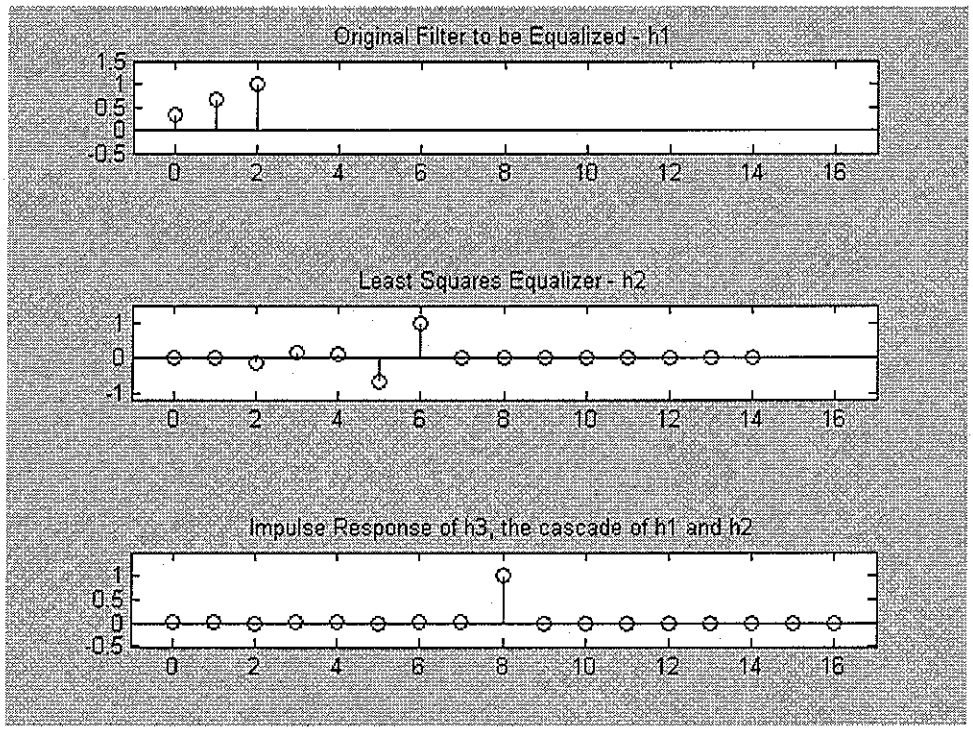

Fig. 6 lsi([1/3 2/3 1],15) - unstable inverse

As an additional example, consider the case where $h_1(n)$ can be thought of as a truncated IIR response. For example, suppose $h_1(n)$ is the sequence {1 1/2 1/4 1/8 1/16 1/32 1/64}. Fig. 7 shows the result of running lsi([1 1/2 1/4 1/8 1/16 1/32 1/64], 12)

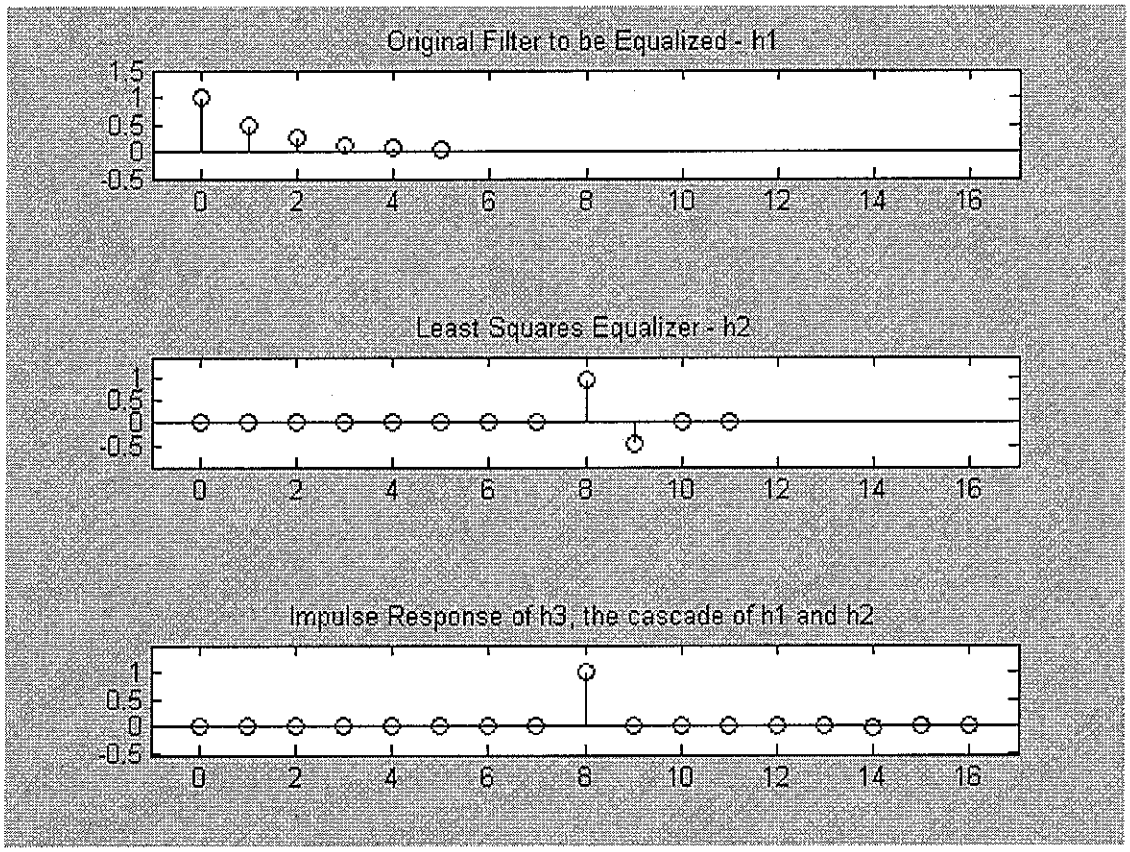

Fig. 7 [h1,h2,h3]=lsi([1 1/2 1/4 1/8 1/16 1/32],12) - truncated IIR

This result is curious in that it has values for $h_2(n)$ that are, for the most part, very close to zero except for two values of approximately 1 and approximately -1/2:

$$h_2 = \{0.0000 \quad 0.0000 \quad -0.0000 \quad 0.0000 \quad 0.0000 \quad 0.0000 \quad 0.0000 \quad -0.0000 \quad 0.9998 \quad -0.4999 \quad -0.0000 \quad 0.0015\} \tag{20}$$

This we recognize at the "right answer" since the sequence $h_1(n)$ for this case is a truncated (to length 6) version of the sequence $(1/2)^n$. The fact that it begins at sample 9 (n=8) is dues to the way the program sets up the target response (r). If we move the 1 in r to the first position (n=0) the response for $h_2(n)$ and $h_3(n)$ shift left accordingly. In any event, it is clear that the FIR equalizer only needs to be length 2 for all practical purposes.

6. A SYSTEM IDENTIFICATION APPROACH

So far we have assumed we know $h_1(n)$ and thus $H_1(z)$. Often times we do, by analysis or by measurement. But we can consider an approach to finding the equalizer that uses a standard system identification (system ID) approach. In this case, we assume that we do not know the system $H_1(z)$, but we do have data giving input samples to $H_1(z)$ and the corresponding output samples. This is more or less a classic "black box" problem.

The approach is to take the output from $H_1(z)$ and find a filter $H_2(z)$ that gives us back the input. We see that this is the inverse filter problem again. In is a system ID problem in the sense that once we find $H_2(z)$ we also have (the unknown) $H_1(z)$. This is further closely related to adaptive filtering and Wiener filtering.

Fig. 8 shows an attempt of invert $H_1(z)$ using a two-tap FIR filter $H_2(z)$. Our goal here is only to set up and solve one specific small problem to achieve a test program. We will use here a notation that is used in adaptive filtering, so the input to the filter $H_1(z)$ is d(n) (desired signal) while the output is x(n) (reference) while the tap weights are denoted by W. Because we want to see if the output of $H_2(z)$, denoted y(n), is the recovered input, we calculate the error e(n). Our goal is to minimize the error, in this case, in the least squares sense.

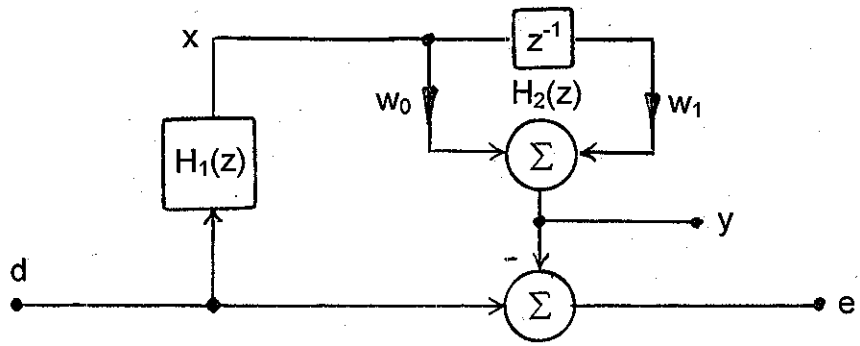

Fig. 8 The system model approach

From Fig. 8 we can write down an expression for e(n):

$$e(n) = d(n) - w_0x(n) - w_1x(n-1) \tag{21}$$

So the squared error on this one sample is:

$$e^2(n) = d^2(n) + w_0^2x(n)^2 + w_1^2x(n-1)^2 \\ -2d(n)x(n)w_0 - 2d(n)x(n-1)w_1 + 2w_0w_1x(n)x(n-1) \tag{22}$$

and the squared error over all samples can be written:

$$E^2 = \sum_n e(n)^2 \tag{23}$$

and this we need to differentiate with respect to w_0 and w_1, setting the partial derivatives to zero:

$$\partial E^2/\partial w_0 = \sum_n \{ (2w_0x(n)^2 - 2d(n)x(n) + 2w_1x(n)x(n-1) \} = 0 \tag{24a}$$

$$\partial E^2/\partial w_1 = \sum_n \{ (2w_1x(n-1)^2 - 2d(n)x(n-1) + 2w_0x(n)x(n-1) \} = 0 \tag{24b}$$

which is, in matrix form:

$$\begin{bmatrix} \sum_n x(n)^2 & \sum_n x(n)x(n-1) \\ \sum_n x(n-1)x(n) & \sum_n x(n-1)^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \sum_n d(n)x(n) \\ \sum_n d(n)x(n-1) \end{bmatrix} \tag{25}$$

Denoting the 2x2 matrix by R (input autocorrelation matrix) and the length 2 column vector on the right by P (cross-correlation vector), the weight vector W is thus:

$$W = R^{-1}P \tag{26}$$

which is usually called the Wiener solution.

Extension of this length 2 solution to a longer weight vector is straightforward [adding x(n-2), x(n-3),.....]. The program wietest.m below allows us to implement and test this approach. In this program example we have first generated a random sequence (d) and then filtered it (x) with our test impulse response h_1 = { 1 2/3 1/3 }. We then choose a length for our FIR equalizer (10 here) and compute the sums over a range of n for the matrix elements of R and for the elements of P. Then the weight vector W is obtained [the impulse response $h_2(n)$] and tested by convolving it with $h_1(n)$. The result depends slightly on the particular run, since the random sequence varies with each run, but for the most part, we see that we get $h_2(n)$ that is in good agreement with the known inverse obtained in Section 2.

```
% wietest.m

d=2*(rand(1,1000)-.5);
x=filter([1 2/3 1/3],1,d);

samps=990
R=zeros(10,10);
for n1=0:9
  for n2=0:9
     for n=1:samps
        R(n1+1,n2+1)=R(n1+1,n2+1)+x(n+n1)*x(n+n2);
     end
  end
end
R
P=zeros(1,10);
for n1=0:9
  for n=1:samps
    P(n1+1)=P(n1+1)+x(n)*d(n+n1);
  end
end
P
W=inv(R)*P'

tst=conv(W,[1 2/3 1/3])
```

Typcial results of running wietest.m are shown below:

W = { 1.0001 -0.6663 0.1101 0.1481 -0.1354
0.0405 0.0180 -0.0245 0.0100 0.0001 } (27)

tst = { 1.0001 0.0005 -0.0008 -0.0006 0.0001 -0.0004 -0.0001
0.0010 -0.0003 -0.0014 0.0034 0.0000 } (28)

Here we see that W, or $h_2(n)$, is very similar to that of Section 2.

7. THE LMS ALGORITHM

Another way to solve the problem is to use the LMS algorithm and have often used a program adapt.m for our demonstrations (below). We should be able to use this program to obtain the Weiner solution. We begin by generating samples exactly as in wietest.m except we make 2000 samples. We then run:

[y,e,w]=adapt(d,x, 14, 0.05)

Fig. 9a shows the first figure of the results of running the test. Here the desired signal is d, and the reference signal is x. The signal x is d filtered by h_1 = { 1 2/3 1/3 }, although this can't be noticed from the figures. Running the program (iterating over all samples, we note that the error gets very small, even by samples 300 or so. This is what we want. When the error becomes effectively zero, the tap weights that are iterated by the "LMS Algorithm:"

$$w_j(n+1) = w_j(n) + 2\mu e(n)x_j(n) \tag{29}$$

must become stationary in time. In this case, the weights w form our FIR equalizer.

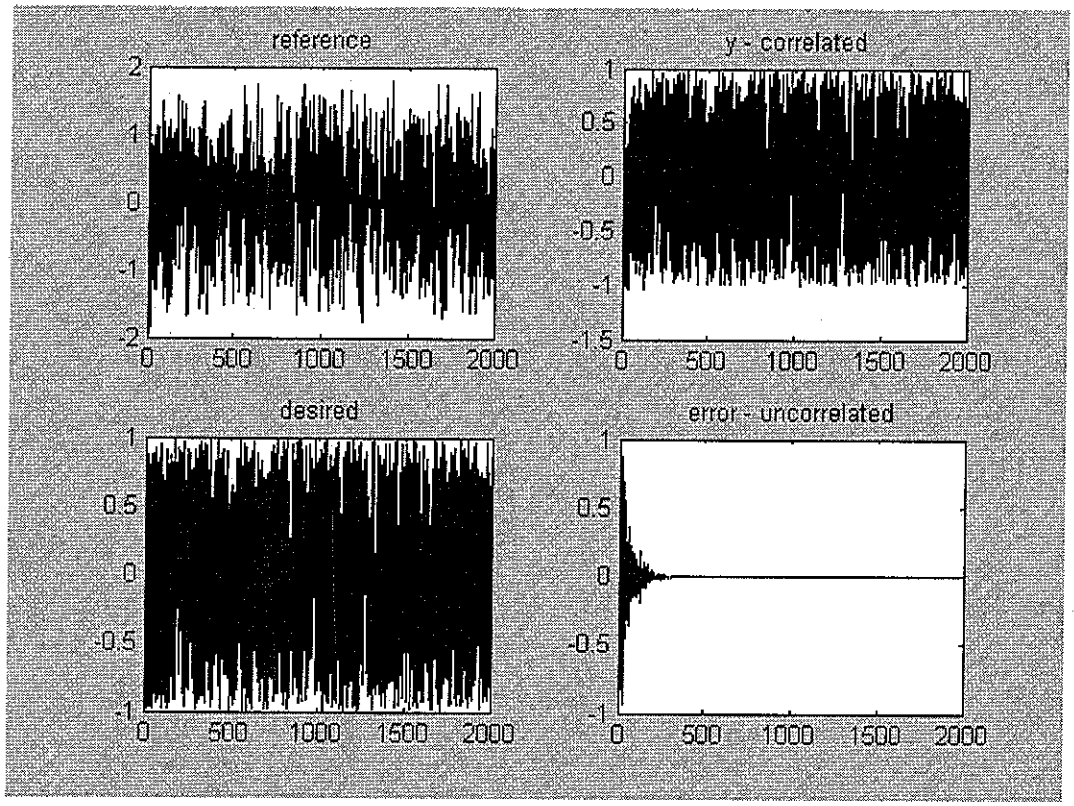

Fig. 9a The results of running adapt.m

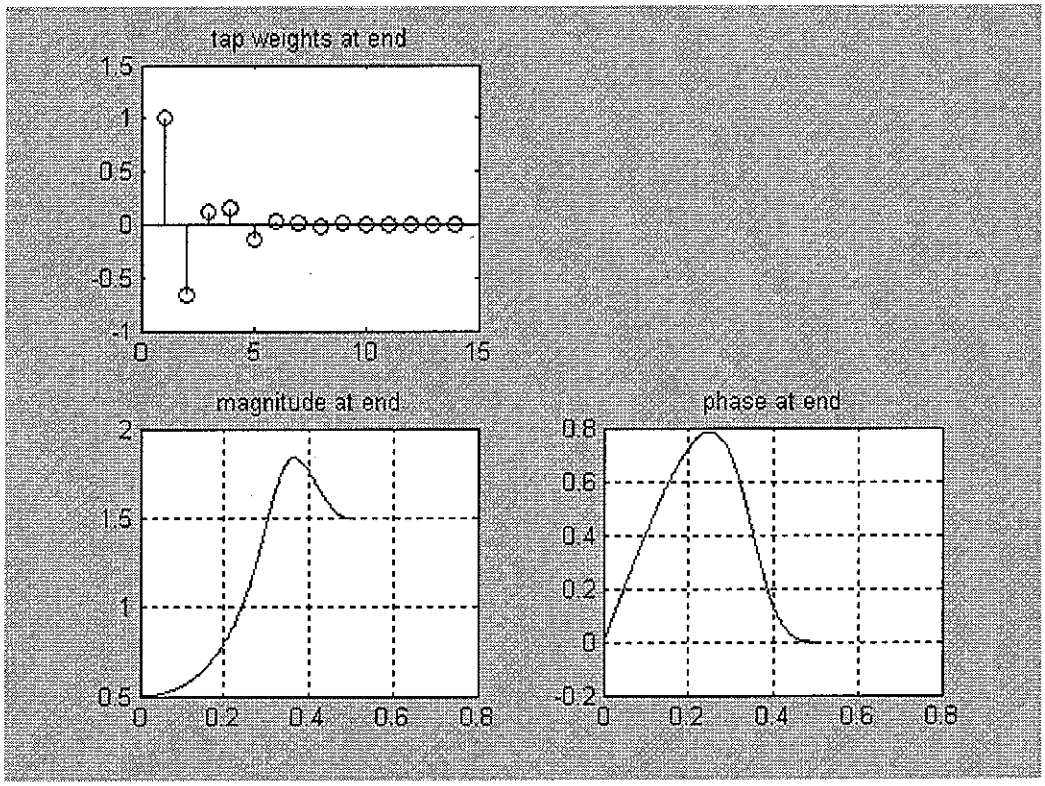

Fig. 9b The tap weights from adapt.m and the frequency response.

Fig. 9b shows the second figure printed by the program, and here of most interest are the now-stationary weights. Printed out, these are for this run:

$$w = \{\ 1.0001\ \ -0.6665\ \ 0.1111\ \ 0.1481\ \ -0.1357\ \ 0.0412\ \ 0.0178$$
$$-0.0256\ \ 0.0112\ \ 0.0010\ \ -0.0043\ \ 0.0025\ \ -0.0003\ \ -0.0003\ \} \qquad (30)$$

By now these are very familiar and can be compared to the results of Section 2 and Section 6.

```
function [y,e,w]=adapt(d,x,L,mu,W0)
%--------------------------------------------------------------------%
%  function [y,e,w]=adapt(d,x,L,mu,W0)                               %
%                                                                    %
%      LMS ALGORITHM ADAPTIVE FILTER SIMULATOR (adapt.m)             %
%                                                                    %
%  INPUTS:                                                           %
%  d is desired signal, x is reference signal, both the same length  %
%  L is the length of the adaptive linear combiner                   %
%  mu is the convergence factor (try about 0.01 to start)            %
%  W0, when included, is an initial weight vector of length L        %
%                                                                    %
%  OUTPUTS:                                                          %
%  y is the output of adaptive linear combiner (correlated signal)   %
%  e is the error (uncorrelated signal)                              %
%  w is the final weight vector                                      %
%--------------------------------------------------------------------%

%--------------------------------------------------------------------%
%                      1         2         3              L-1       %
%      x            -----     -----     -----            -----      %
%      ___________ |  -1 |___|  -1 |___|  -1 |___    ___|  -1 |__    %
%            \     | z   | | | z   | | | z   | |       | z   |  |    %
%             \    -----  |  -----  |  -----  |       -----   |    %
%              \ W0    W1 \        / W2      / W3             /     %
%               \          -----------      /               /      %
%                \-------|    SUM    |----/           <-----/      %
%                        -----------                               %
%                             |                                    %
%                             |--------------------------- y       %
%                          (-)|                                    %
%                       (+)  -----                                 %
%  d --------------------------| SUM |------------------------ e    %
%                              -----                               %
%                                                                  %
%     LMS Algorithm    Wj(n+1) = Wj(n) + 2*mu*e(n)*xj(n)           %
%                                                                  %
%        B Hutchins     1993  2001   2006                         %
%--------------------------------------------------------------------%


if exist('W0')==1 w=W0;    % check for user-given initial weights
else w=zeros(1,L);         % else set taps to zero
end

xj=zeros(1,L);             % initialize linear combiner delay line

for n = 1:length(d)            %  for each iteration
     xj=[x(n),xj(1:L-1)];      %  move data up line
     y(n)=xj*w';               %  compute y
     e(n)=d(n)-y(n);           %  compute e
     w=w+2*mu*e(n).*xj;        %  update taps w(n+1)=w(n)+2mux(n)e(n)
end                            %
```

```
figure(1);
clf                              %  plot the four signals
subplot(221)                     %
plot(x)                          %  x
title('reference')               %
subplot(222)
plot(y)                          %  y
title('y - correlated')          %
subplot(223)
plot(d)                          %  d
title('desired')                 %
subplot(224)
plot(e)                          %  e
title('error - uncorrelated')    %


figure(2)                        %  now plot the impulse and freq. resp.
clf
subplot(221)                     %  of the final taps.  This is strictly
stem(w)                          %  valid information only if the error
title('tap weights at end')      %  is zero.   If the error is very
subplot(223)                     %  small, and/or if mu is very small,
H=freqz(w,1,500);                %  the result may be valid of thought
plot([0:.001:.499],abs(H))       %  of as an instantaneous frequency
grid                             %  response.
title('magnitude at end')        %
subplot(224)
plot([0:.001:.499],angle(H))
grid
title('phase at end')
figure(2)
```