LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-824190

Signal Preconditioning to Minimize Impulse Response Contribution

S. K. Lehman

July 7, 2021

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Signal Preconditioning to Minimize Impulse Response Contribution

Sean K. Lehman

July 07, 2021

Abstract

A study was performed to identify a method to minimize the effect of a linear time-invariant (LTI) system impulse response on an input. Three methods were studied: Wiener filter, the N4SID algorithm and transfer function estimation, the latter two using functions from MATLAB's System Identification Toolbox. Although all three methods were able estimate an unknown forward impulse response given an input/output time series pair, only the Wiener filter was able to estimate a system inverse which satisfactorily solved the problem using a cosine similarity measure.

LLNL-TR-824190

Auspices This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Contents

1 Introduction

All input/output systems have transfer functions which modify the input. For example, wave-based transducers emit the derivative (high-pass filtered version) of their inputs, at a minimum. A study was performed to identify a method to minimize the effect of a linear time-invariant (LTI) system impulse response on an input. That is, given the LTI system,

$$y \;=\; h * x, \tag{1}$$

where x_n is the input time series, y_n the output and h_n the forward system impulse response, determine a preconditioned input time series, x_{pre} such that

$$y \;\approx\; h * x_{\mathrm{pre}}. \tag{2}$$

In theory, $x_{\mathrm{pre}} = h^{-1} * x$. Only the input and output time series are available, the system impulse response, h, is unknown. Its inverse must be estimated.

Three algorithms were studied,

- The Wiener filter (see Appendix A);
- The MATLAB numerical algorithm for state-space identification function, `n4sid()`;
- The MATLAB transfer function estimation function, `tfest()`.

The latter two functions are from the System Identification Toolbox. Although determining an estimate of the impulse response, $\hat{h}$, is not required, the algorithms were used to identify it as a confirmation they functioned as expected on an input/output test set.

All three methods were successful at approximating the forward impulse response in proof-of-concept test using a 1–12 Hz chirp input and a single cycle 6.5 Hz sinusoid impulse response. However, only the Wiener filter estimation of the inverse was successful at compensating for the forward impulse response's effect on the input. The measure of success is the cosine similarity as described in Appendix B.

The next section presents an overview of signal preconditioning. Section 3 develops a proof-of-concept. Section 4 presents the empirical results using data from an electromechanical transducer system.

2 Signal Preconditioning

As shown in Figure 1, signal preconditioning involves convolving the input time series, x_n, with an approximation to the inverse impulse response, $\hat{h}^{-1}$, resulting in a preconditioned time series, $x_{\rm pre}$, which minimizes the effect of the forward impulse response, resulting in a close approximation to the input signal at the output. This is more easily understood in the frequency domain where convolution transforms into a multiplication by the transfer function,

$$\hat{X} = H\left(H^{-1}X\right) = HX_{\rm pre}, \tag{3}$$

where $X_{\rm pre} \equiv H^{-1}X$ is the preconditioned signal spectrum.

The inverse impulse response is estimated by swapping the input/output time series, as shown in Figure 2, in the impulse response estimation algorithms.

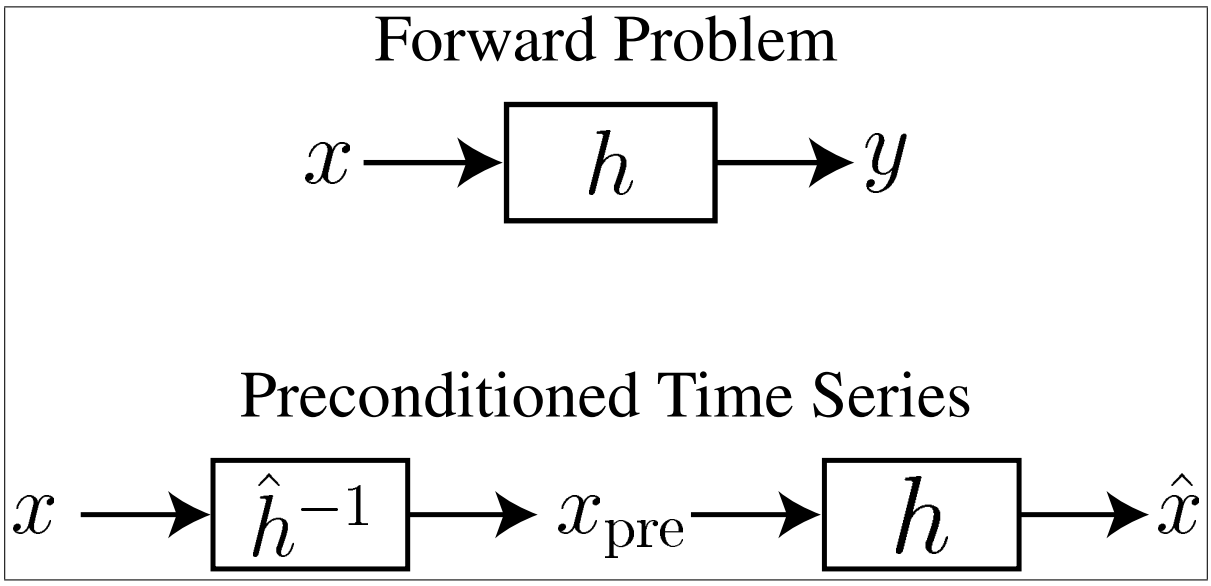

Figure 1: Preconditioned problem. Given an input/output time series, x and y, with unknown system impulse response, h, estimate the inverse impulse response, $\hat{h}^{-1}$. Then precondition the input so that when presented to the forward system, its output closely matches the original input. That is, convolve x with an approximation to the inverse impulse response, $\hat{h}^{-1}$, such that the effects of the impulse response, h, are minimized.

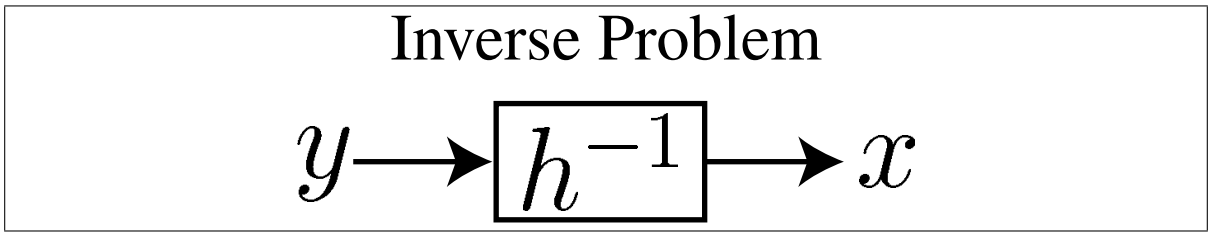

Figure 2: Inverse problem. The inverse impulse response may be estimated by swapping the input/output time series in the algorithms.

3 Proof-of-Concept

The proof-of-concept used a 1–12 Hz chirp as input and a 6.5 Hz single cycle sinusoid as forward impulse response. The time series are presented in Figure 3. The upper left plot presents the input, x. The upper right shows the output, y, with peak around 6.5 Hz. The middle left plot shows the true impulse response in red along with the Wiener, state-space ("SS", `n4sid()`) and transfer function ("TF", `tfest()`) estimates. The true impulse response is 24 samples. In a real situation where the impulse response length is unknown, a guess must be made of its length. In this example, 73 samples was selected. Similarly with the state-space and transfer function estimates. A value of 2 states was selected for the former, and 6 poles and 7 zeros for the latter.

The middle right plot presents the estimated inverse impulse response for the three methods where the number of Wiener taps remains 73 but the states were increased to 6 and the poles and zeros to 12 and 13, respectively. The state-space and transfer function numbers were selected in an attempt to maximize the cosine similarity between the chirp input and preconditioned output. However, given the poor inverse performance of the the `n4sid()` and `tfest()` functions, the task proved futile.

The bottom left graph shows the preconditioned input using the Wiener approach, the graph to its right overlays the chirp input and the preconditioned output. The similarity is 0.982 indicating a good match and the success of the Wiener algorithm.

Table 1 lists the simulation parameters and results. The similarity measures reflect the success in the forward problem for the three methods and the poor performance with the inverse with the state-space and transfer function methods.

For completeness, Figure 4 presents the state-space and transfer function inverse results. Evidently, for this proof-of-concept problem, the System Identification Toolbox functions perform well and as expected in the forward but not inverse case. When examining the spectra presented in the bottom plot of Figure 5, one observes good matches with the chirp input, yet their time domain equivalents have poor similarity measures. This indicates there is an issue with the phases. Perhaps, this may be explored at another time.

Table 1: Proof-of-concept parameters.

Simulation Parameters	
Input, x, & output, y, size	1441 samples
Forward impulse response, h, size	24 samples
Forward Wiener filter taps	73 samples
Inverse Wiener filter taps	73 samples
Forward state-space (SS) size	2
Forward transfer function (TF)	6 poles / 7 zeros
Inverse state-space (SS) size	6
Inverse transfer (TF)	12 poles / 13 zeros
Sample interval	6.94e-03 seconds
Regularization, alpha	1.00e-12
SNR	40 dB
Noise variance	6.11e-02
Similarity Measure between Input & Preconditioned Output	
Input, x, & output, $\hat{x}$	0.982
Similarity Measures between Forward & Estimated Impulse Responses	
True & Wiener estimate	0.990
True & SS estimated	0.899
True & TF estimate	0.998
Similarity Measures between Input and Preconditioned Output	
Input & Wiener estimate	0.982
Input & SS estimate	-0.127
Input & TF estimate	0.676

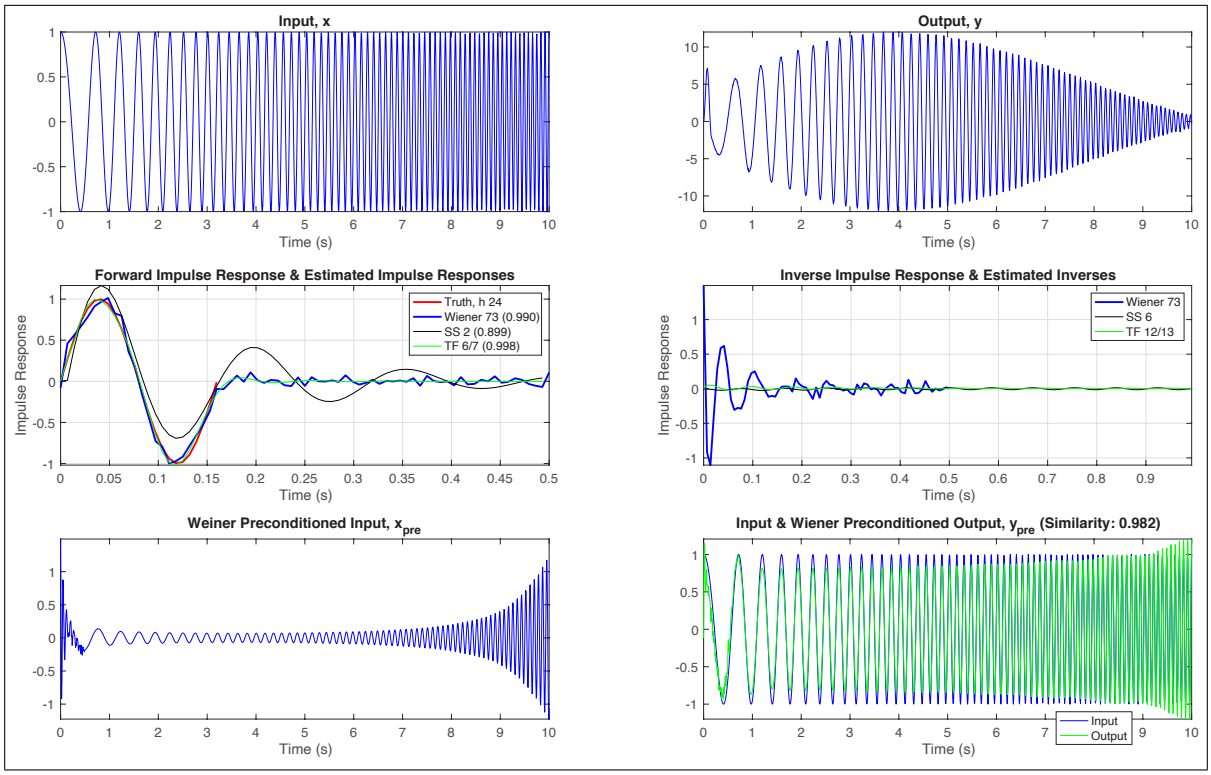

Figure 3: Proof-of-concept results with the Wiener filter algorithm.

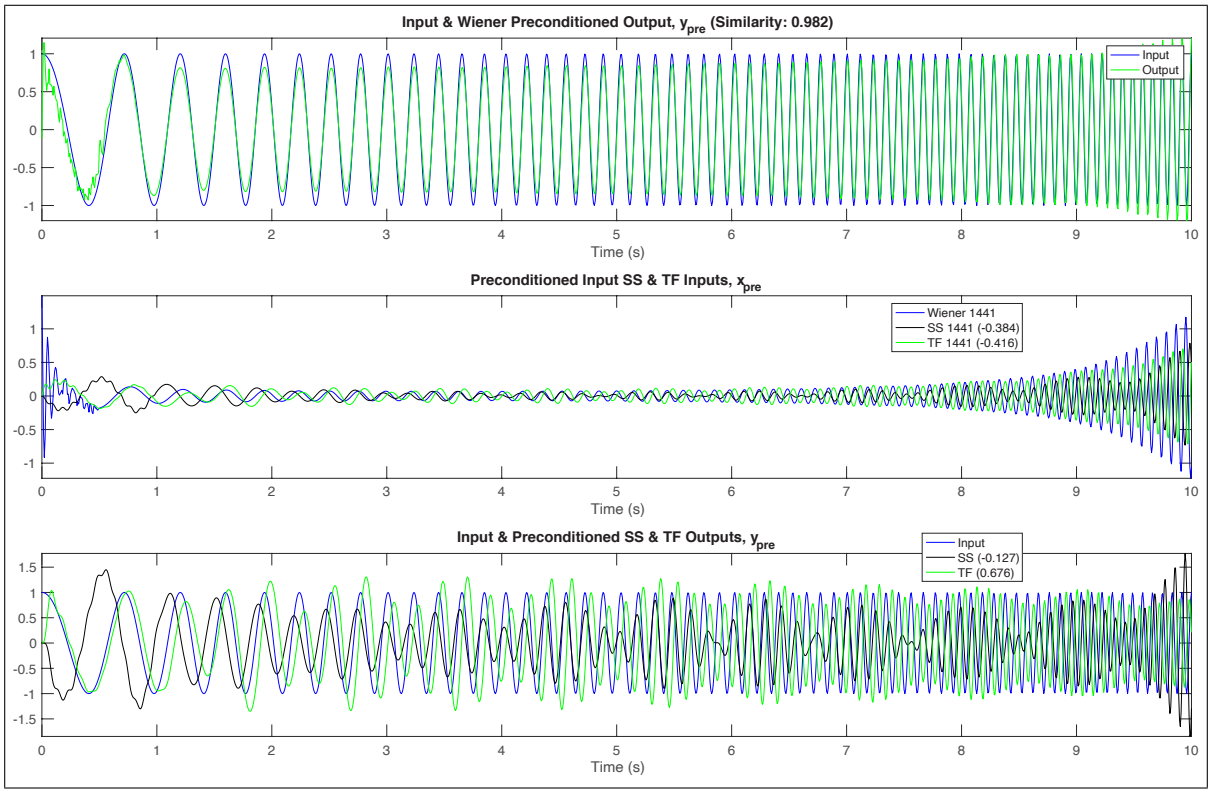

Figure 4: Proof-of-concept results with the state-space and transfer function algorithms.

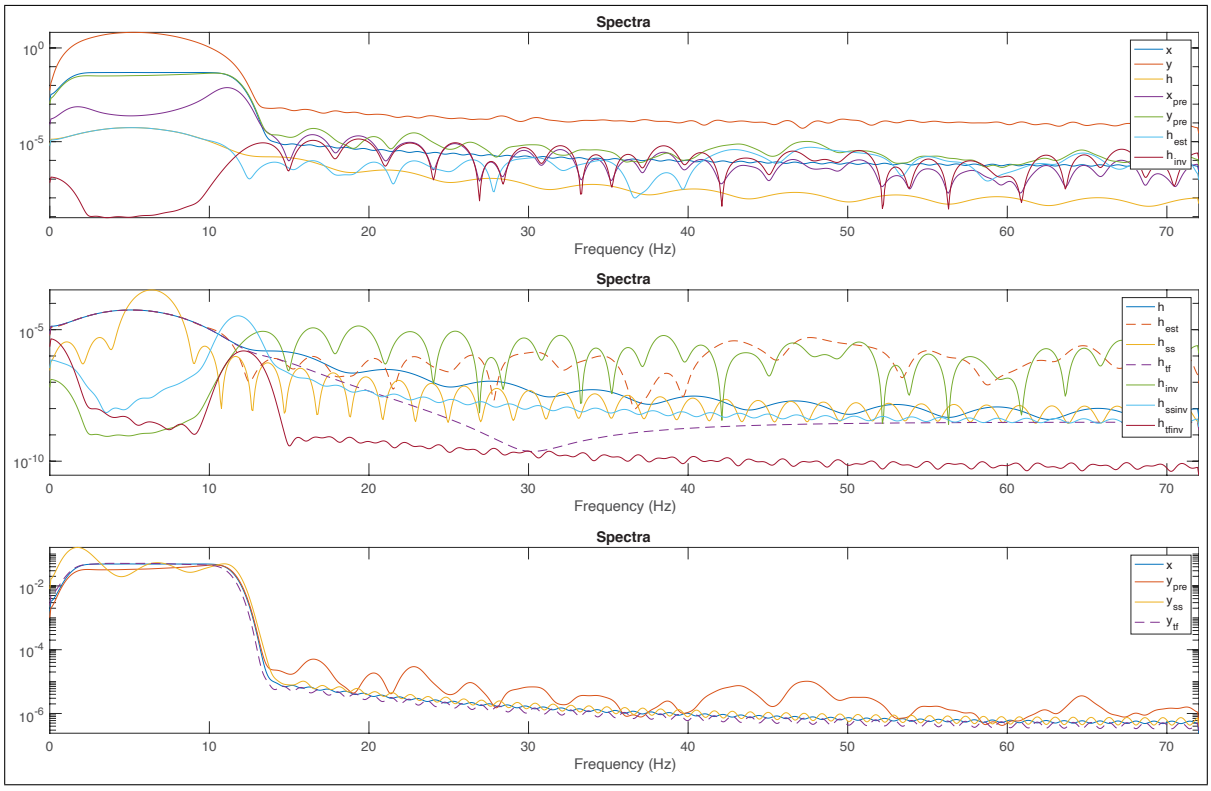

Figure 5: Proof-of-concept spectral results. The input, original output and impulse response are represented by x, y and h respectively. x_{pre} and y_{pre} are the Wiener filter based preconditioned input and resulting output. h_{est} and h_{inv} are the Wiener based forward and inverse impulse responses. h_{ss} & h_{tf} are the state-space and transfer function estimates. h_{ssinv} & h_{tfinv} are the corresponding inverse estimates. y_{ss} & y_{tf} are the state-space and transfer function preconditioned outputs. It is curious their spectra are good matches with the chirp input, yet their time domain equivalents have poor similarity measures. This indicates there is an issue with the phases.

4 Empirical Results

The Wiener filter algorithm was applied to a real transducer system in which an input voltage signal drives an electromechanical transducer. The goal is to have the input signal match the output motion. Two data sets were collected. The first, presented in Figure 6, was subjectively quieter and required no regularization in the Wiener filter. A maximum similarity of 0.97 between the input and output time series was achieved by iterating over the number of filter taps. The result was 2500 points.

The second data set, shown in Figure 7, was noisier. It required a regularization constant of 10^{-3}. A maximum similarity of 0.87 was achieved using 8192 filter points.

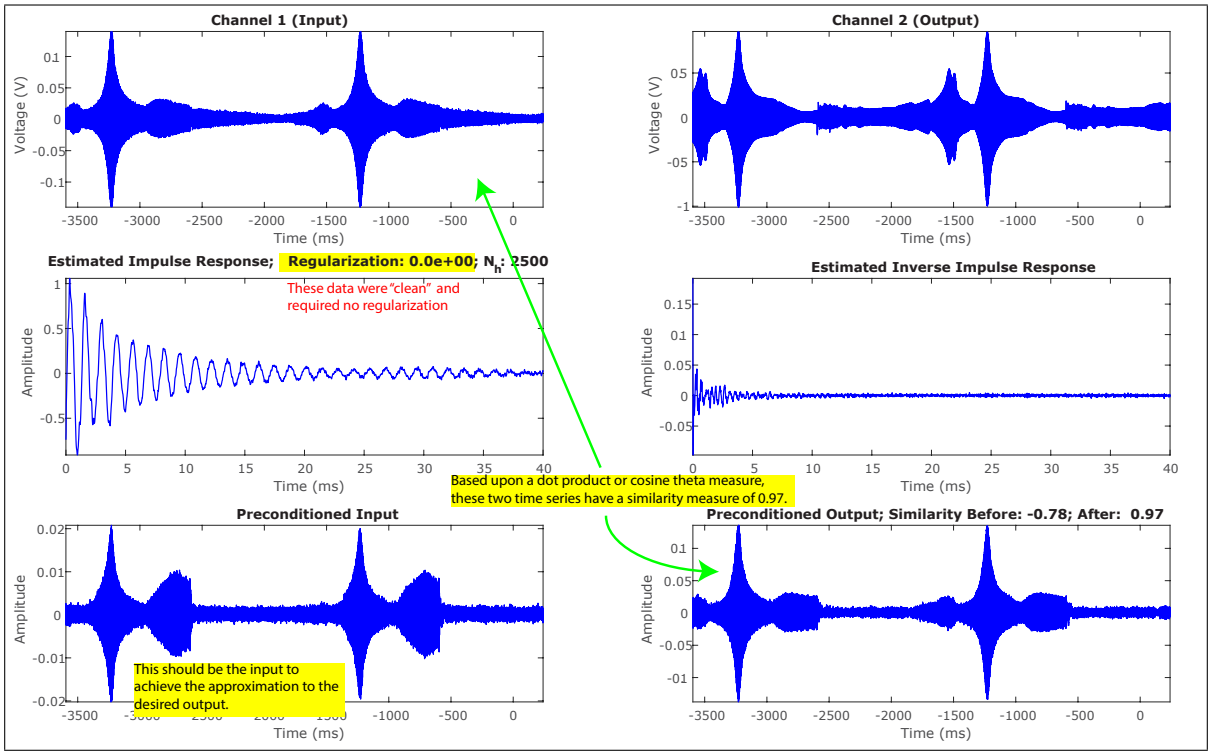

Figure 6: First input/output time series.

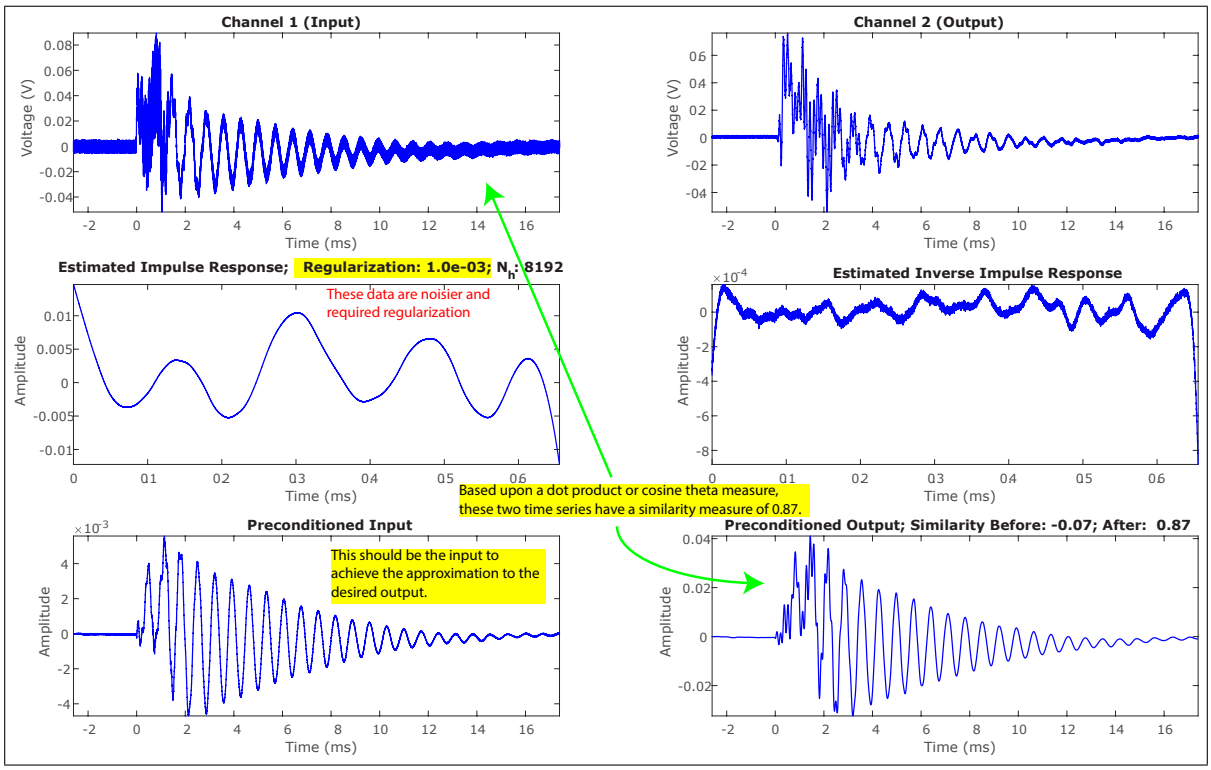

Figure 7: Second input/output time series.

5 Conclusion

Three methods were evaluated to solve a signal preconditioning problem:

- The Wiener filter (see Appendix A),
- The MATLAB numerical algorithm for state-space identification function, `n4sid()`,
- The MATLAB transfer function estimation function, `tfest()`,

the latter two functions having been taken from the MATLAB System Identification Toolbox.

Although all three algorithms solved the forward problem, only the Wiener filter successfully solved the inverse and preconditioning problem. Further investigation is required to determine why the state-space and transfer function methods failed at the inverse problem.

A Wiener Filter

There are multiple interpretations of the Wiener filter. A deconvolution interpretation is used here.

Consider a linear time-invariant system, shown in Figure 8, described by

$$y_n = h_n * x_n, \tag{4}$$

$$y_n = \sum_{m=0}^{N-1} h_m x_{n-m} \tag{5}$$

where x_n is the input, y_n is the output and h_n is the unknown system impulse response. The Wiener solution is to find h_n such that

$$h_n = \underset{h_n}{\text{argmin}}\, \mathrm{E}\left\{e_n^2\right\}, \tag{6}$$

where

$$e_n \equiv y_n - \sum_{m=0}^{N-1} h_m x_{n-m}. \tag{7}$$

Expand $\mathrm{E}\{e_n^2\}$,

$$\mathrm{E}\left\{e_n^2\right\} = \mathrm{E}\left\{\left(y_n - \sum_{m=0}^{N-1} h_m x_{n-m}\right)^2\right\}, \tag{8}$$

$$= \mathrm{E}\left\{y_n^2\right\} + \mathrm{E}\left\{\left(\sum_{m=0}^{N-1} h_m x_{n-m}\right)^2\right\} - 2\mathrm{E}\left\{\sum_{m=0}^{N-1} h_m x_{n-m}\, y_n\right\}. \tag{9}$$

Minimize with respect to h_k,

$$\frac{\partial}{\partial h_k}\mathrm{E}\left\{e_n^2\right\} = 2\mathrm{E}\left\{\sum_{m=0}^{N-1} h_m x_{n-m}\, x_{n-k}\right\} - 2\mathrm{E}\left\{x_{n-k}\, y_n\right\}, \tag{10}$$

$$= 2\sum_{m=0}^{N-1} h_m \mathrm{E}\left\{x_{n-m}\, x_{n-k}\right\} - 2\mathrm{E}\left\{x_{n-k}\, y_n\right\}, \tag{11}$$

$$= 2\sum_{m=0}^{N-1} h_m \mathrm{R}_{xx}(m-k) - 2\mathrm{R}_{yx}(k) \equiv 0, \tag{12}$$

where the correlations are defined as

$$\mathrm{R}_{xx}(m) = \mathrm{E}\left\{x_n\, x_{n-m}\right\}, \tag{13}$$

$$\mathrm{R}_{yx}(k) = \mathrm{E}\left\{x_{n-k}\, y_n\right\}. \tag{14}$$

Eqn. (12) is a matrix equation,

$$\mathrm{R}_{xx}(m-k) \cdot \mathbf{h}(m) = \mathrm{R}_{yx}(k) \tag{15}$$

with solution,

$$\mathbf{h}(m) \;=\; \mathrm{R}_{xx}^{-1}(m-k)\cdot \mathrm{R}_{yx}(k). \tag{16}$$

When the data are noisy, a regularization may be included with the inverse as,

$$\mathbf{h}(m) \;=\; \left(\mathrm{R}_{xx}(m-k)+\alpha\right)^{-1}\cdot \mathrm{R}_{yx}(k), \tag{17}$$

where α is a positive constant.

Given input, x_n, and output, y_n, time series, Eqn. (16) is used to identify the *forward* impulse response, h_n.

B Similarity Measure

An inner product or $\cos\theta$ test is used to determine the similarity between to functions or time series in particular,

$$m \;=\; \frac{y\cdot x}{|y|\,|x|}. \tag{18}$$

Using this measure, identical time series yield $m \equiv 1$, time series which are 180 degrees out of phase have $m \equiv -1$ and those which are completely dissimilar result in $m \equiv 0$.

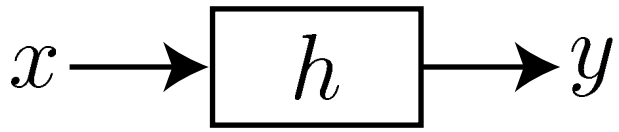

Figure 8: Forward linear time-invariant problem.

C MATLAB Levinson-Wiggins-Robinson Algorithm, `lwr.m`

The Levinson-Wiggins-Robinson (lwr) algorithm is the core of the Wiener filter. It solves Eqn. (16) or generally Eqn. (17) when there is a need for a regularization constant.

```
function [h,varargout] = lwr( x , y , Norder , varargin )
%*************************************************************************
%
% TITLE:    lwr.m
% AUTHOR:   Sean K. Lehman
% DATE:     April 08, 2021
% FUNCTION: This is the Levinson-Wiggins-Robinson algorithm to compute the
%           FIR Wiener solution.
%           This is based upon JVC's original lwr.m code.
%
%              y = h * x;
% SYNTAX:
%                   h = lwr( x , y , Norder );
%              [h,Rxx] = lwr( x , y , Norder );
%          [h,Rxx,Ryx] = lwr( x , y , Norder );
%                  ... = lwr( x , y , Norder , alpha );
%
%            x  is the input.
%            y  is the output.
%            h  is the impulse response to be estimated.
%        Norder is the number of FIR taps or fiter order.
%        alpha  is an optional regularization parameter.
%
% CALLS:
%
% MODIFICATIONS:
%
%
%*************************************************************************%
alpha  = 0;
switch nargin
    case 4
        if ¬isempty(varargin{1}); alpha = varargin{1}; end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JVC's lwr() code returns a result which matches MATLAB's firwiender()
% code.  His code uses FD_Xcorr() which normalizes by,
%     N    = max( Nx , Ny );
%     Nfft = 2^nextpow2(N);
%     if Nfft/2 < N; Nfft = 2*Nfft; end
% Apply the same normalization here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normalization
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N               = max([ length(x) length(y) ]);
Nfft            = 2^nextpow2( N );
if Nfft/2 < N; Nfft = 2*Nfft; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Correlations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[Rxx,lags]      =  xcorr(x(:),  Norder);
Ryx             =  xcorr(y(:),x(:),Norder);
Rxx             = Rxx / Nfft;
Ryx             = Ryx / Nfft;
n               =  find( lags==0 );
ndx             =  (0:Norder)';
Rxx             =  Rxx(n+ndx);
Ryx             =  Ryx(n+ndx);
Rxx(1)          =  Rxx(1) + alpha;
a               =  zeros([Norder   1]);
k               =  zeros([Norder   1]);
kstar           =  k;
h               =  zeros([Norder+1 1]);
a(1)            =  1;
Δ           =  Rxx(2);
p               =  Rxx(1);
kstar(1)        =  Δ/p;
a(2)            = -kstar(1);
h(1)            =  Ryx(1)/p;
del             =  h(1)*Rxx(2);
p               =  p - kstar(1)*Δ;
k(1)            =  (del-Ryx(2))/p;
h(1)            =  h(1) - k(1)*a(2);
h(2)            = -k(1);
if Norder+1 ≥ 3
    for n = 3:Norder+1
        Δ        = a(1:n-1)'*Rxx(n:-1:2);
        kstar(n-1)   = Δ/p;
        at           = [a(1:n-1); 0];
        a            = at-kstar(n-1)*flipud(at);
        del          = h(1:n-1)'*Rxx(n:-1:2);
        p            = p-kstar(n-1)*Δ;
        k(n-1)       = (del-Ryx(n))/p;
        h(1:n)       = [h(1:n-1); 0] - k(n-1)*a(n:-1:1);
    end
end

switch nargout
    case 2
        varargout{1} = Rxx;
    case 3
        varargout{1} = Rxx;
        varargout{2} = Ryx;
end
```

D MATLAB Wiener-Based Signal Preconditioner Algorithm

`Precondition.m`

```
function [xpre,varargout] = Precondition(x,y,varargin)
%*************************************************************************
%
% TITLE:    Precondition.m
% AUTHOR:   Sean K. Lehman
% DATE:     April 19, 2021
% FUNCTION: Use a Wiener filter to attempt to precondition an input to
%           minimize the effects of a transfer function
% SYNTAX:             xpre = Precondition(x,y)
%                     xpre = Precondition(x,y,N)
%                     xpre = Precondition(x,y,N,alpha)
%                     xpre = Precondition(x,y,N,alpha,method)
%               [xpre,hinv] = Precondition(...)
%            [xpre,hinv,h] = Precondition(...)
%
%           method is either 'lwr' or 'firwiener'
% CALLS:
%
% MODIFICATIONS:
%
%
%*************************************************************************
x      = reshape( x , [] , 1 );
y      = reshape( y , [] , 1 );
Nx     = length( x );
alpha  = 0;
Method = 'lwr';
if Nx ≠ length(y)
    error('Inputs must have the same lengths');
end
Nh     = fix( Nx/2 );

switch nargin
    case 3
        if ¬isempty(varargin{1}); Nh = varargin{1}; end
    case 4
        if ¬isempty(varargin{1}); Nh = varargin{1}; end
        if ¬isempty(varargin{2}); alpha = varargin{2}; end
    case 5
        if ¬isempty(varargin{1}); Nh = varargin{1}; end
        if ¬isempty(varargin{2}); alpha = varargin{2}; end
        Method = varargin{3};
end

switch Method
```

```
    case 'lwr'
        hinv = lwr( y , x , Nh , alpha );
    case 'firwiener'
        hinv = firwiener( Nh , y , x )';
    otherwise
        error('%s:  Method is either ''lwr'' or ''firwiener'''',Method);
end

xpre = conv( hinv , x ) ;
xpre = xpre(1:Nx);

%           [xpre,hinv]   = Precondition(...)
%           [xpre,hinv,h] = Precondition(...)
switch nargout
    case 2
        varargout{1} = hinv;
    case 3
        varargout{1} = hinv;
        switch Method
            case 'lwr'
                varargout{2} = lwr( x , y , Nh , alpha );
            case 'firwiener'
                varargout{2} = firwiener( Nh , x , y )';
        end
end
```

E MATLAB M-File Which Demonstrates the Preconditioner `tst_Precondition.m`

```
function tst_Precondition(varargin)
%*************************************************************************
%
% TITLE:    tst_Precondition.m
% AUTHOR:   Sean K. Lehman
% DATE:     June 14, 2021
% FUNCTION: Test the Precondition.m function
% SYNTAX:
% CALLS:    Precondition()
%           fetchrigure()
%           Similarity()
%
%
% QUESTION:
% Given an LTI of the form,
%           y = conv( h , x )
%
% Seerk a preconditioned input, x_{pre}, to minimize the difference
% between y_{pre} and x based upon a cosine similarity measure.
%
% That is, seek an input such that
%          m = y_{pre}' * x / (norm(y_{pre}) * norm(x))
% is maximized, where
%          y_{pre} = conv( h , x_{pre} )
% and
%          x_{pre} = conv( h^{-1} , x )
% and h^{-1} is an estimate of the inverse of h.
%
% As demonstrated in this M-file, a Wiener approach using MATLAB's
% firwiener() function yielded the best results.
%
% However, an approach using the System Identification Toolbox functions,
% n4sid() and tfest() is also tried.
%
% Although they estimate the forward impulse response well, they failed
% miserably at estimating the inverses for preconditioning.
%
%
%
%*************************************************************************
Print = 'no';
switch nargin
    case 1
        Print = varargin{ 1 };
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define the figure aspect ratio.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
AspectRatio = [11  8.5]/4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form a chirp input signal, x.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flim   = [ 1 12 ];
tend   = 10;
dt     = 1/(12*max(flim));
t      = (0:dt:tend)';
x      = chirp(t,flim(1),tend,flim(2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get the signal length.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nt     = length( t );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create a single cycle sine impulse response.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Fc     = 6.25;
th     = (0:dt:1/Fc)';
Nhimp  = length( th );
h      = sin( 2*pi*Fc*th );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filter the input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y      = conv( h , x );
y      = y( 1:Nt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Add noise.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SNR        = 40; % dB
SigmaNoise = std(y) * 10^(-SNR/20);
noise      = SigmaNoise * randn([Nt 1]);
y          = y + noise;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Given the input/output data, determine the filter using the Wiener
% approach.
% Guess at Nid since, in practice, the length of the impulse response, h,
% is unknown.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nid    = 3 * Nhimp;
% hest   = firwiener( Nid , x , y )';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the inverse.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% hinv   = firwiener( Nid , y , x )';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the predistored input signal.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xpre   = conv( hinv , x ) ;
```

```
% xpre   = xpre( 1:Nt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the preconditioned input signal using the Precondition() function
% which implements all of the above steps.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha            = 1e-12;
[xpre,hinv,hest] = Precondition(x,y,Nid,alpha,'lwr');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the output using the predistored signal.
% This should be a close match to the input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ypre     = conv( hest , xpre ) ;
ypre     = ypre( 1:Nt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Use the ID Toolbox.
% Put the time series into ID toolbox format.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dforward  = iddata( y , x , dt ); % For the forward estimate
dinverse  = iddata( x , y , dt ); % For the inverse estimate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimate the forward SS & TF models using the ID toolbox.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nss       = 2;
Ntf       = 6;
ssopt     = n4sidOptions( 'EnforceStability' , true );
ssforward = n4sid( dforward , Nss , ssopt );
tfforward = tfest( dforward , Ntf );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimate the inverse models.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nssinv    = 3 * Nss;
Ntfinv    = 2 * Ntf;
ssinverse = n4sid( dinverse , Nssinv , ssopt );
tfinverse = tfest( dinverse , Ntfinv );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create impulse for impulse responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u         = zeros([2*Nid 1]);
u(1)      = 1;
u         = iddata( [] , u , dt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the forward impulse responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ipssforward   = sim( ssforward , u(1:Nid) );
iptfforward   = sim( tfforward , u(1:Nid) );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the inverse impulse responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ipssinverse   = sim( ssinverse , u );
```

```
iptfinverse   = sim( tfinverse , u );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get the estimated impulse response time series.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hss    = get(ipssforward,'y');
htf    = get(iptfforward,'y');
hssinv = get(ipssinverse,'y');
htfinv = get(iptfinverse,'y');

% hss    = impulse( ssforward );
% htf    = impulse( tfforward );
% hssinv = impulse( ssinverse );
% htfinv = impulse( tfinverse );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the predistored input signals.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpress   = conv( hssinv , x ) ;
xpress   = xpress( 1:Nt );
xpretf   = conv( htfinv , x ) ;
xpretf   = xpretf( 1:Nt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the output using the predistored signals.
% This should be a close match to the input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ypress   = conv( hss , xpress ) ;
ypress   = ypress( 1:Nt );
ypretf   = conv( htf , xpretf ) ;
ypretf   = ypretf( 1:Nt );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the similarity between the Wiener predistored output and the
% desired output.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
similarity = Similarity( ypre , x );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the similarity of the forward impulse response estimates.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hpadded          = zeros( [Nid+1 1] );
hpadded(1:Nhimp) = h;
hsimilarity      = [ ...
    Similarity( hpadded        , hest )
    Similarity( hpadded(1:Nid) , hss  )
    Similarity( hpadded(1:Nid) , htf  )
    ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Save the figure handles.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figureList = [0;0;0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fetch the figure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figureList(1) = fetchfigure( 'Signal Preconditioning I' , AspectRatio );clf
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(321)
plot( t , x , 'b' );
xlabel( 'Time (s)' );
title( 'Input, x' );
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the output.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(322)
plot( t , y , 'b' );
xlabel( 'Time (s)' );
title( 'Output, y' );
axis tight
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the forward impulse responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(323)
hp = plot( ...
    t(1:size(h))  , h    , 'r' , ...
    t(1:Nid+1)    , hest , 'b' , ...
    t(1:Nid)      , hss  , 'k' , ...
    t(1:Nid)      , htf  , 'g' , ...
    'linewidth' , 1);
set( hp(1:2) , 'linewidth' , 2 );
axis tight
grid on
xlabel('Time (s)');
ylabel('Impulse Response');
title('Forward Impulse Response & Estimated Impulse Responses');
legend( ...
    sprintf('Truth, h %d',Nhimp) , ...
    sprintf('Wiener %d (%.3f)',Nid+1,hsimilarity(1)) , ...
    sprintf('SS %d (%.3f)',Nss,hsimilarity(2)) , ...
    sprintf('TF %d/%d (%.3f)',Ntf+[0 1],hsimilarity(3)) );
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the inverse impulse responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(324)
% hp = plot( ...
%     t(1:Nid+1)    , hinv   , 'b' , ...
%     t(1:Nid)      , hssinv , 'k' , ...
%     t(1:Nid)      , htfinv , 'g' , ...
%     'linewidth' , 1);
hp = plot( ...
    t(1:Nid+1)              , hinv   , 'b' , ...
    (0:length(hssinv)-1)*dt , hssinv , 'k' , ...
```

```
250     (0:length(htfinv)-1)*dt , htfinv , 'g' , ...
251     'linewidth' , 1);
252 set( hp(1) , 'linewidth' , 2 );
253 axis tight
254 grid on
255 xlabel('Time (s)');
256 ylabel('Impulse Response');
257 title('Inverse Impulse Response & Estimated Inverses');
258 legend( ...
259     sprintf('Wiener %d',Nid+1), ...
260     sprintf('SS %d',Nssinv), ...
261     sprintf('TF %d/%d',Ntfinv+[0 1]));
262 set(gca,'fontsize',14);
263 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
264 % Plot the Wiener predistorted input.
265 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
266 subplot(325)
267 plot( ...
268     t , xpre , 'b', ...
269     'linewidth' , 1);
270 axis tight
271 xlabel('Time (s)');
272 title('Weiner Preconditioned Input, x_{pre}');
273 set(gca,'fontsize',14);
274 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
275 % Plot the Wiener predistorted output.
276 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
277 subplot(326)
278 plot( ...
279     t , x    , 'b' , ...
280     t , ypre , 'g' , ...
281     'linewidth' , 1);
282 axis tight
283 xlabel('Time (s)');
284 title( sprintf('Input & Wiener Preconditioned Output, y_{pre} ...
        (Similarity: %.3f)',similarity) )
285 legend('Input','Output','Location','NorthEast')
286 set(gca,'fontsize',14);
287 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
288 % Fetch the figure.
289 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
290 figureList(2) = fetchfigure( 'Signal Preconditioning II' , AspectRatio ...
        );clf
291 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
292 % Re-plot the Wiener predistorted output.
293 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
294 subplot(311)
295 plot( ...
296     t , x    , 'b' , ...
297     t , ypre , 'g' , ...
298     'linewidth' , 1);
```

```
axis tight
xlabel('Time (s)');
title( sprintf('Input & Wiener Preconditioned Output, y_{pre} ...
    (Similarity: %.3f)',similarity) )
legend('Input','Output','Location','NorthEast')
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the SS & TF predistorted inputs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(312)
plot( ...
    t , xpre   , 'b', ...
    t , xpress , 'k', ...
    t , xpretf , 'g', ...
    'linewidth' , 1);
axis tight
xlabel('Time (s)');
title('Preconditioned Input SS & TF Inputs, x_{pre}');
legend(...
    sprintf('Wiener %d',length(xpre)) , ...
    sprintf('SS %d (%.3f)',length(xpress),Similarity(xpre,xpress)) , ...
    sprintf('TF %d (%.3f)',length(xpretf),Similarity(xpre,xpretf)) );
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the SS & TF predistorted outputs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(313)
plot( ...
    t , x      , 'b' , ...
    t , ypress , 'k' , ...
    t , ypretf , 'g' , ...
    'linewidth' , 1);
axis tight
xlabel('Time (s)');
title( 'Input & Preconditioned SS & TF Outputs, y_{pre}' );
legend(...
    'Input',...
    sprintf('SS (%.3f)',Similarity(xpress,x)),...
    sprintf('TF (%.3f)',Similarity(xpretf,x)));
set(gca,'fontsize',14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute spectra
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Nwindow  = 128;
Noverlap = Nwindow / 2;
Nfft     = 2^nextpow2( Nt );
[X,freq] = pwelch( x , Nwindow , Noverlap , Nfft , 1/dt );
Y        = pwelch( y , Nwindow , Noverlap , Nfft , 1/dt );
H        = pwelch( [h;zeros([Nt-Nhimp 1])] , Nwindow , Noverlap , Nfft ...
    , 1/dt );
```

```
HEST     = pwelch( [hest;zeros([Nt-length(hest) 1])] , Nwindow , ...
    Noverlap , Nfft , 1/dt );
XPRE     = pwelch( xpre , Nwindow , Noverlap , Nfft , 1/dt );
YPRE     = pwelch( ypre , Nwindow , Noverlap , Nfft , 1/dt );
YPRESS   = pwelch( ypress , Nwindow , Noverlap , Nfft , 1/dt );
YPRETF   = pwelch( ypretf , Nwindow , Noverlap , Nfft , 1/dt );
HSS      = pwelch( [hss;zeros([Nt-length(hss) 1])] , Nwindow , Noverlap ...
    , Nfft , 1/dt );
HTF      = pwelch( [htf;zeros([Nt-length(hss) 1])] , Nwindow , Noverlap ...
    , Nfft , 1/dt );
HINV     = pwelch( [hinv;zeros([Nt-length(hinv) 1])] , Nwindow , ...
    Noverlap , Nfft , 1/dt );
HSSINV   = pwelch( [hssinv;zeros([Nt-length(hssinv) 1])] , Nwindow , ...
    Noverlap , Nfft , 1/dt );
HTFINV   = pwelch( [htfinv;zeros([Nt-length(htfinv) 1])] , Nwindow , ...
    Noverlap , Nfft , 1/dt );
% S = [ X Y H HEST XPRE YPRE YPRESS YPRETF HSS HTF HINV HSSINV HTFINV ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fetch the spectra figure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figureList(3) = fetchfigure( 'Signal Preconditioning III' , AspectRatio ...
    );clf
subplot(311)
semilogy( ...
    freq , [ X Y H XPRE YPRE HEST HINV ], ...
    'linewidth' , 1 );
xlabel( 'Frequency (Hz)' );
title( 'Spectra' );
legend('x','y','h','x_{pre}','y_{pre}','h_{est}','h_{inv}');
axis tight
set( gca , 'fontsize' , 14 );

subplot(312)
hp = semilogy( ...
    freq , [ H HEST HSS HTF HINV HSSINV HTFINV ] , 'linewidth' , 1 );
set( hp([2 4]) , 'linestyle' , '--' );
xlabel( 'Frequency (Hz)' );
title( 'Spectra' );
legend('h','h_{est}','h_{ss}','h_{tf}','h_{inv}','h_{ssinv}','h_{tfinv}');
axis tight
set( gca , 'fontsize' , 14 );

subplot(313)
hp = semilogy( ...
    freq , [ X YPRE YPRESS YPRETF ] , 'linewidth' , 1 );
set( hp(4) , 'linestyle' , '--' );
xlabel( 'Frequency (Hz)' );
title( 'Spectra' );
legend('x','y_{pre}','y_{ss}','y_{tf}');
axis tight
set( gca , 'fontsize' , 14 );
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Print useful information
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(1,'%25s : %6d\n','x & y size, Nt',Nt);
fprintf(1,'%25s : %6d\n','h size, Nhimp',Nhimp);
fprintf(1,'%25s : %6d\n','Wiener size, Nid',Nid);
fprintf(1,'%25s : %6d\n','hest size, Nid+1',Nid+1);
fprintf(1,'%25s : %6d\n','hinv size, Nid+1',length(hinv));
fprintf(1,'%25s : %6d\n','Forward SS size',Nss);
fprintf(1,'%25s : %6d/%d\n','Forward TF size',Ntf+[0 1]);
fprintf(1,'%25s : %6d\n','Inverse SS size',Nssinv);
fprintf(1,'%25s : %6d/%d\n','Inverse TF size',Ntfinv+[0 1]);
fprintf(1,'%25s : % .3f\n','x, Wiener similarity',similarity);
fprintf(1,'%25s : % .3f\n','x, SS similarity',Similarity(xpress,x));
fprintf(1,'%25s : % .3f\n','x, TF similarity',Similarity(xpretf,x));
fprintf(1,'%25s : % .3f\n','h, Wiener similarity',hsimilarity(1));
fprintf(1,'%25s : % .3f\n','h, SS similarity',hsimilarity(2));
fprintf(1,'%25s : % .3f\n','h, TF similarity',hsimilarity(3));
fprintf(1,'%25s : %.2e seconds\n','Sample interval, dt',dt);
fprintf(1,'%25s : %.2e\n','Regularization, alpha',alpha);
fprintf(1,'%25s : %g dB\n','SNR',SNR);
fprintf(1,'%25s : %.2e\n','Noise variance',SigmaNoise);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Print figures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if strcmp( Print , 'yes' )
    fprintf(1,'   Printing figures ... ');
    for n = 1:length(figureList)
        EPSFile = sprintf('Signal_Precondition-%d.eps',n);
        fprintf(1,'%s ... ',EPSFile);
        figure( figureList(n) );
        orient landscape;
        print( EPSFile , '-depsc2' );
    end
    fprintf(1,'done.\n');
end



end % function tst_Precondition()

function s = Similarity(x,y,varargin)
%**************************************************************************
%
% TITLE:    Similarity.m
% AUTHOR:   Sean K. Lehman
% DATE:     April 19, 2021
% FUNCTION: Measure the similarity between two time series
% SYNTAX:
%           s = Similarity(x,y)
%           s = Similarity(x,y,Measure)
```

```
%
%                 Measure is 'cosine' (default)
%                            'numsim'
%  where the similarity between two numbers, 'numsim' is
%               numsim(a,b) = 1 - abs( a - b ) / (abs(a)+abs(b))
% CALLS:
%
% MODIFICATIONS:
%
%
%************************************************************************
Measure = 'cosine';
switch nargin
    case 3, Measure = varargin{ 1 };
end

x = reshape( x , [] , 1 );
y = reshape( y , [] , 1 );

if length(x) ~= length(y)
    error('Inputs must have the same lengths');
end

switch Measure
    case 'numsim'
        s = mean( 1 - abs( x-y ) ./ (abs(x) + abs(y)) );
    otherwise % cosine measure
        s = y' * x / (norm(y) * norm(x));
end
end % s = Similarity(x,y,varargin)


function fig = fetchfigure( FigureName , varargin )
%************************************************************************
%
% TITLE:    fetchfigure.m
% AUTHOR:   Sean K. Lehman
% DATE:     July 08, 2004
% FUNCTION: Fetch a figure by name
%
% SYNTAX:   fig = fetchfigure( FigureName )
%           fig = fetchfigure( FigureName , scale )
%
% MODIFICATIONS: Philip Top 11/24/09 modified function to use findobj
%  method,  should be slightly faster than previous methodology
%
%
%************************************************************************/

fig = findobj(get(0,'Children'),'Name',FigureName);
if isempty(fig) % Create new window
```

```
494      scrnsize = get( 0 , 'screenSize' );
495      fig = figure('Name',FigureName,'visible','off');
496      pos = get( fig , 'position' );
497
498      switch length(varargin)
499          case 1
500              scale = varargin{1};
501              if length(scale)==1; scale = scale*[1 1]; end
502          otherwise
503              scale = [1 1];
504      end
505
506      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
507      % Center on screen
508      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
509      pos(1) = (scrnsize(3)-scale(1)*pos(3))/2;
510      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
511      % Because the Windows OS is a cluster f**k, one must correct for the
512      % vertical mis-position.
513      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
514      if ispc
515          pos(2) = (scrnsize(4)-scale(2)*pos(4))/2;
516      end
517      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
518      % Set the figure position
519      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
520      set( fig , 'position' , [pos(1) pos(2) scale(1)*pos(3) ...
             scale(2)*pos(4)] );
521  end
522  figure(fig);
523  end % fig = fetchfigure( FigureName , varargin )
```