LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
-LIGO-
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUE OF TECHNOLOGY

Technical Note	LIGO-T130074-v1	September 25, 2013

Simulating the Advanced LIGO Interferometer Using the Real Control Code

Juan F. Castillo, The University of Texas at El Paso

California Institute of Technology
LIGO Project –MS 18-34
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project –NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 1970
Mail Stop S9-02
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

https://www.ligo.caltech.edu/

1 Abstract

The interferometer of the advanced LIGO requires complicated software controls to operate. To better understand both the hardware and software, we should apply the same software used to control the interferometers in a simplified simulation. This model will incorporate a simulation of the control code to the internal seismic isolation system of the horizontal access module (HAM ISI). By using the actual control code, the simulation model will expose issues specific to that code and distinguish them more easily from issues that may arise in the hardware of the interferometers. The model has shown successful dampening results when applied to the undamped system. The ability to troubleshoot the overall system by using a simulation is critical in effectively dealing with present and future problems.

2 Introduction

The goal of the Advanced Laser Interferometer Gravitational Observatory is to conclusively identify gravitational waves from cosmological events. By using a Michelson interferometer, the system will directly observe the difference between the path lengths of two perpendicular laser beams [1]. The LIGO Livingston and Hansford sites are currently undergoing a major renovation. New hardware systems are being equipped on site in order to yield far superior results than the previous installation [2]. While some components are still being updated, other components, such as the Dual Recycle Michelson Interferometer (DRMI), are being tested for functionality and stability. Advanced LIGO is moving into a commissioning phase, which takes the newly installed hardware and performs adjustments meant to maximize performance.

The measurements being taken from the system are extraordinarily small. Extreme caution must be taken to ensure the functionality of the physical instruments as well the operational software. Having a series of simulations that can predict the behavior of hardware as well as software coding could vastly improve the success of future adjustments [3]. Implementing a real-time simulation of the hardware that communicates to the control code will generate a practical troubleshooting technique. A simulation model operating on a nightly basis, continuously verifying that changes to the control code work, will create a viable diagnostic tool.

3 APPROACH

The objective of this project is to implement a real-time simulation of the HAM ISI control code to the cavity and suspension models of the interferometer. The HAM ISI defines what the analog system of interferometer senses. The system takes controls signals from the actuator and transmits this information to the sensors [1]. The project focused on two topics, the schematic and the noise filters, to create the simulation. Matlab Simulink was used to build the schematic and Matlab scripts were used to synthesize the noise filters.

Simulink is used as a drawing tool whose output is parsed by a script to generate a C code. The actual C code is used to run the simulation. Simulink is merely used as a drawing program [4]. The figure below shows the simulation model from the top level. The yellow highlighted box represents the simulation component and the pink component represents the library part identical to the control code of the HAM ISI.

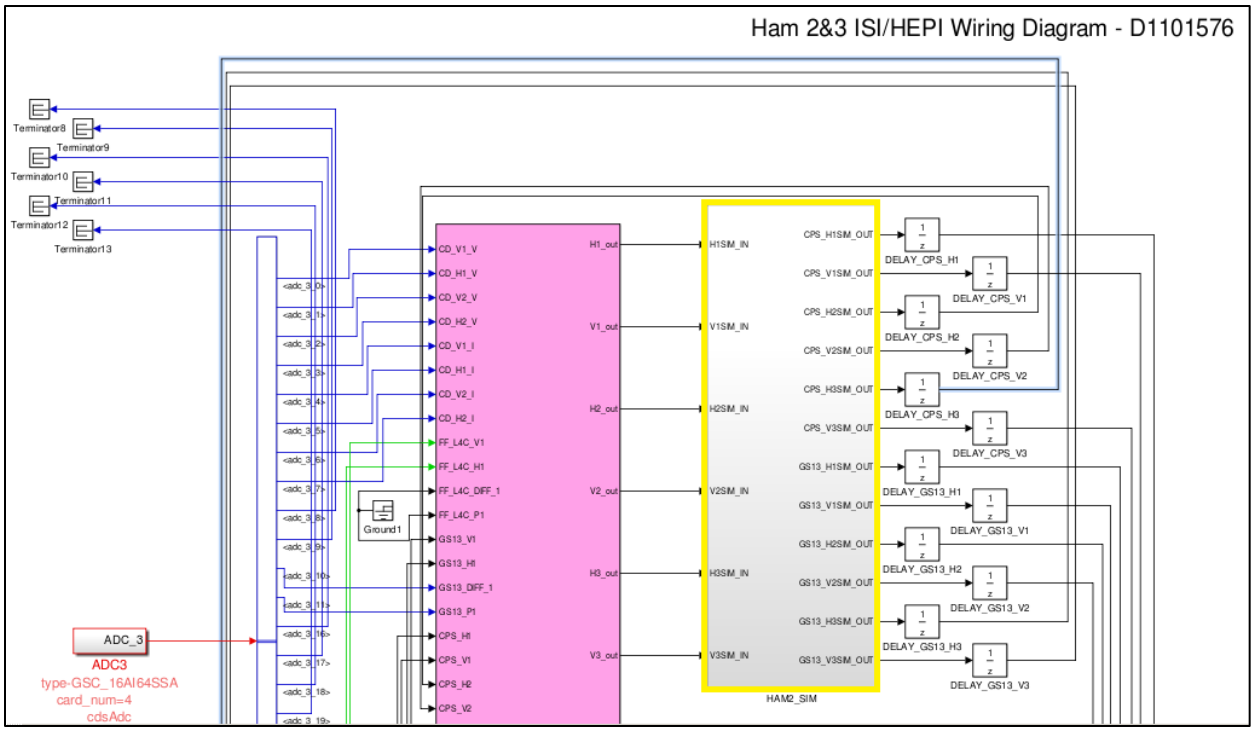

Figure 1: Top Level Simulink schematic

The simulation's output is the control code's input and the control's output is the simulation's input. In order to replicate a real-time interaction between the two components, noise filters were incorporated.

Understanding what noises the interferometer contends with is critical in reproducing the systems reactions. Noise is anything that makes it harder to measure the gravitational wave signal in the interferometer [1]. These noises affect the stabilization of the components in relation to each other. The main focus of this simulation is seismic noise. Seismic noise has many forms. Major factors range from earthquakes to the ocean tides to basic human traffic [5]. There are many other noises such as shot noise and electronic noise that are not represented. Filter components are used to simulate the noise in the model.

Through a series of Matlab scripts and functions, the components are embedded into the simulation's schematic. Two filter arrays were built using this process, the actuator to sensor filters and the ground to sensor filters. The output of both arrays are added together, creating a single response signal for each sensor. The signals are ultimately received by the HAM ISI control code. The graph below represents one of the filter's frequencies from the actuator to a CPS (position) sensor. The script achieved results that produce an accurate representation of the discrete data into a state-space representation.

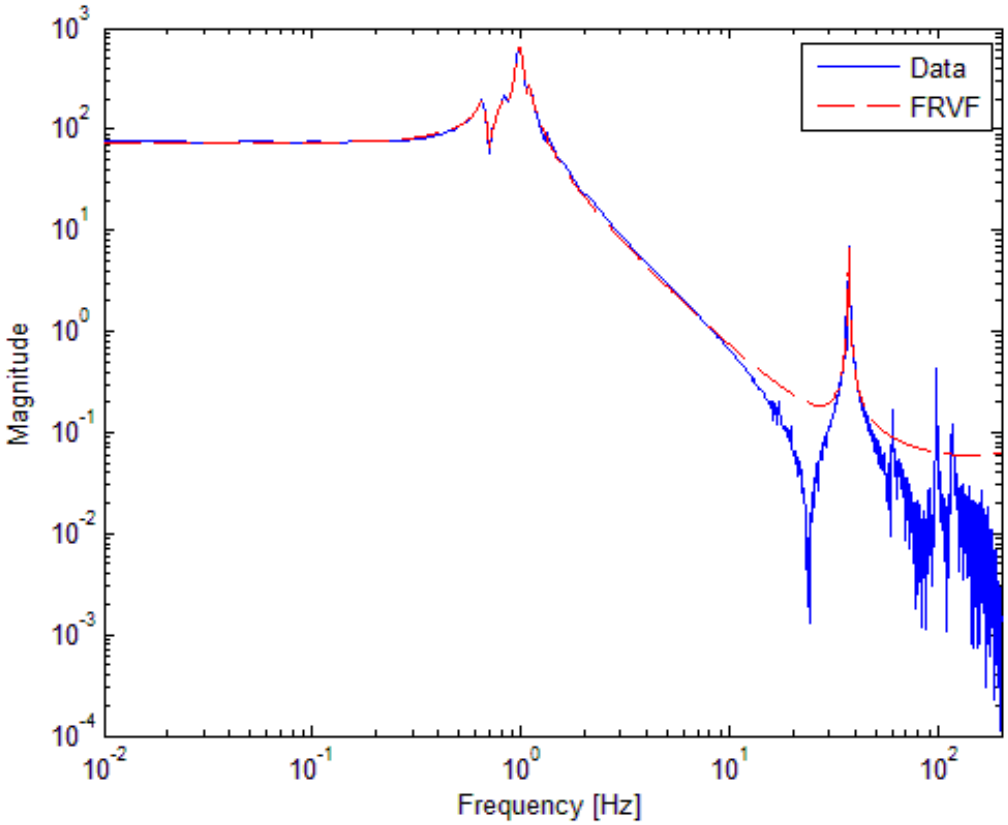

Figure 2: Magnitude Plot - blue is actual data and red is approximation

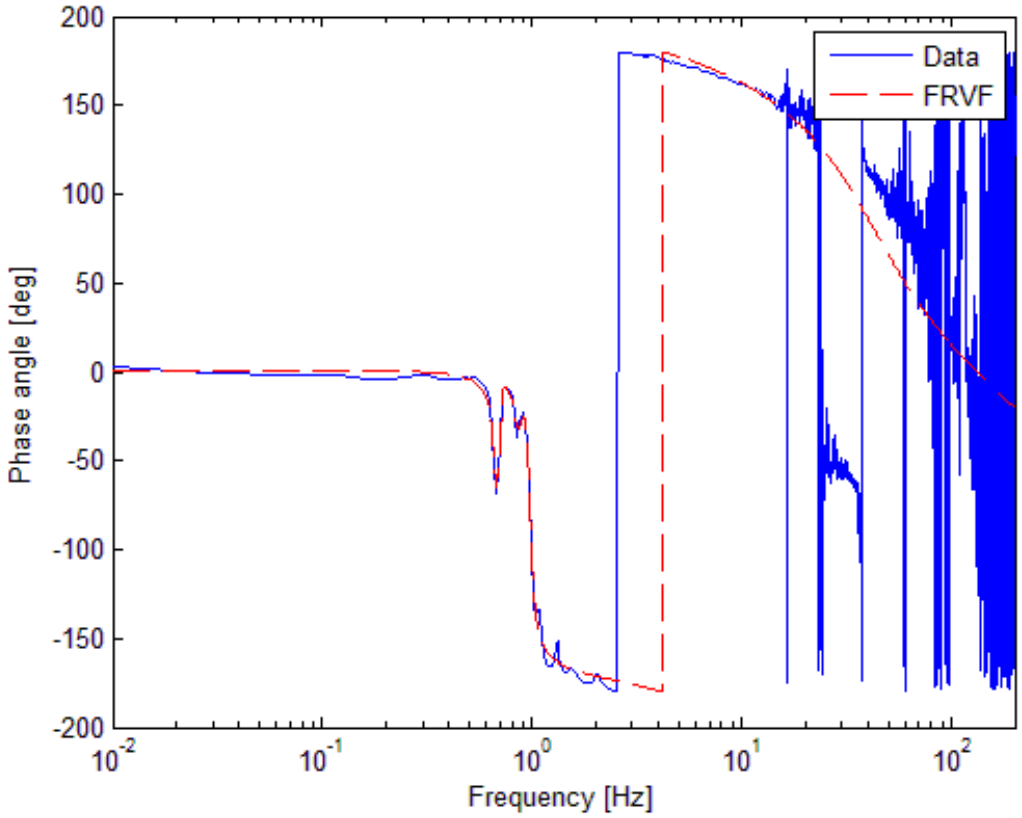

Figure 3: Phase Plot - blue is actual data and red is approximation

There are some additional calibrations that had to be included into the simulation. The HAM ISI control code component outputs its signals in actuator basis. Therefore, the data from the control code cannot be transmitted to the simulation component without some calibration. One inverted matrix was added to convert the simulation's input data from actuator basis to the Cartesian basis. The output signals are then recalibrated from Cartesian basis back to sensor basis using two inverted matrices, one for each sensor.

An additional calibration had to be implemented as well. The simulation component was designed to handle actual measurement units, but the HAM ISI's data is digitized in counts. Thus, a set of inverse filters were created to calibrate the incoming digital signals from counts to actual measurements units. This process was reversed and applied to the outgoing signals. Figure 4 shows the internal components of the simulation block. For more specifics and information regarding this process, please refer the Section 5 titled Methods.

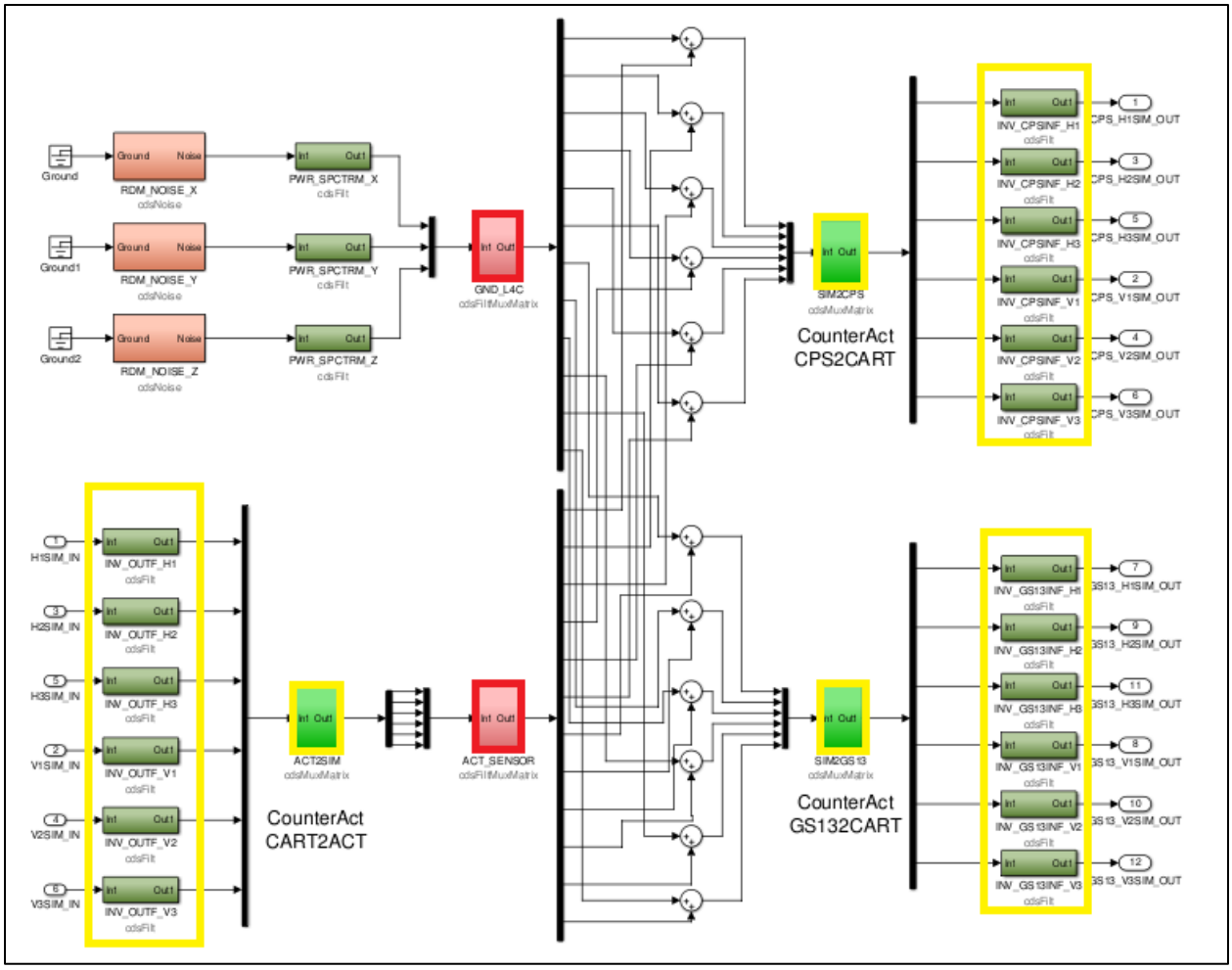

Figure 4: Inside the Simulation Block

The yellow highlighted green boxes are the inverted matrices used as basis converters. The yellow highlighted olive boxes are the inverted filters used for measurement calibration.

4 CONCLUSIONS

The simulation model has provided some successful results. The model has been compiled and was loaded onto the test stand at LIGO. It is fully operational as a real-time system and is currently gathering data. We have been able to prove that the model is able to dampen the system in an undamped state. Figure 5 shows the results from a graph of the power spectrum.

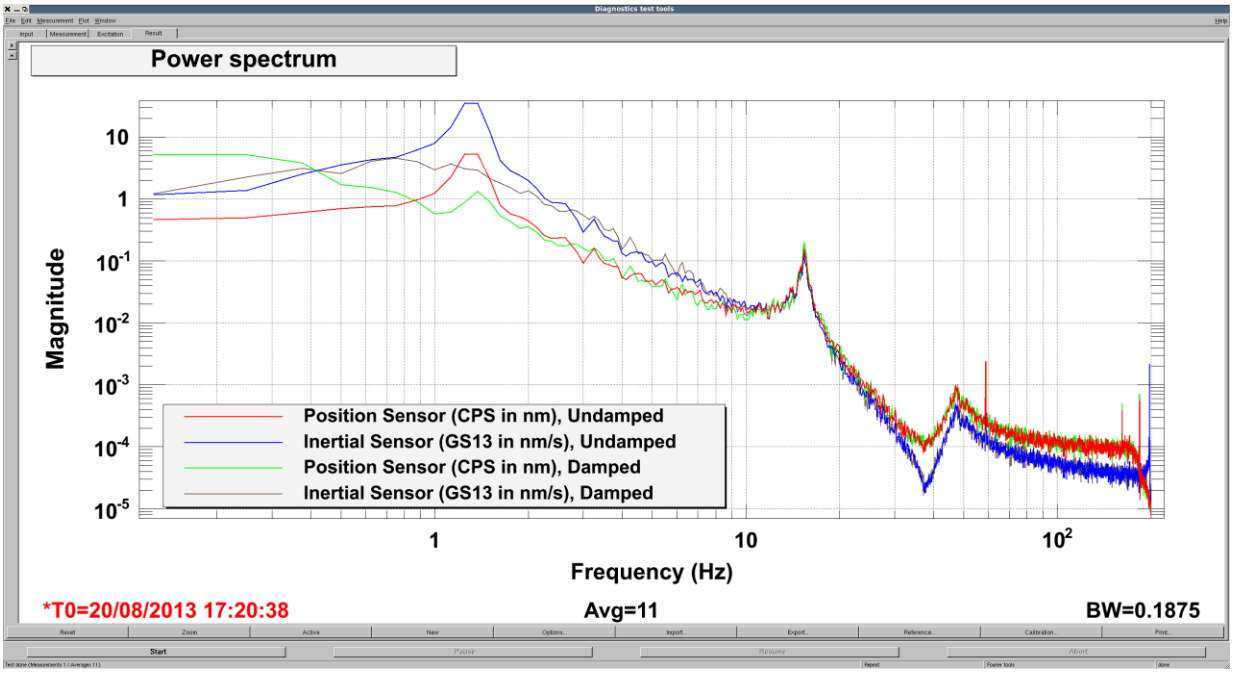

Figure 5: Power Spectrum Graph

The red line is the undamped CPS position sensor in the HAM ISI. When the damping filters are turned on, the result is graphed as a green line. At approximately 1 Hz, the magnitude of the power spectrum is effectively damped. The same results are shown for the GS13 inertial sensor by comparing the undamped blue line with the damped gray line. This graph provides evidence that the actual controls damp the simulated ground noise in the same way that the controls damp the real ground noise. The graph also shows how the model needs improvement in some critical areas.

At low frequencies below 1 Hz, the damped results show a higher magnitude than undamped results. This is in stark contrast to actual data models where the lines are level with each other. We have concluded that there are issues with the calibration of the data at low frequencies. We have explored various possibilities, such as the physical units in the Cartesian basis not matching. Unfortunately we have not been able to debug the simulation. Although the model is not without error, I am confident we have laid the groundwork for future modifications to be incorporated in an overall effective simulator.

There are also a number of possible expansions stemming from this project. Eventually, noises that were neglected should be incorporated such as the shot and electronic noise. A

simulation of the suspension system should also be included for the overall model. As a side note, to verify the model's reliability, an alternative approach for building a simulation can be applied. Instead of using the data model fitting approach we created, a model using first principals (size and mass of objects, etc. with relation to the physical world) could be created. Using our model with a first principle model could establish proof of trustworthiness.

5 METHODS

5.1 MATLAB SIMULATION MODEL

The control code component defines what the analog system of the interferometer senses and then transmits this data using actuators and sensors The types of sensors we focused on are the CPS (capacitive position sensor) and GS13 (Geotech Seismic) sensors, both of which are analog. The CPS is a position sensor and the GS13 is an inertial sensor. These sensors carry information pertaining to the degrees of freedom in the seismic isolation system. The analog data from these sensors is digitized using an analog to digital converter. The digital data is then used by the suspension and cavity system to correct abnormities in position due to noise. Using a large variety of filters helps to distinguish what events are causing seismic noise. The system can then reestablish its overall orientation by using precision actuators that align the components.

After the adjustments mentioned above have corrected the orientation of the cavity and suspension, the resulting data is looped back into the control code. This data, which is in digital form, is sent through a digital to analog converter and transmitted into the interferometer to start the process over. This established sequence creates a real-time control code process that will operate indefinitely.

5.2 FILTER DEFINITION

The filters are created using several Matlab scripts listed in the Appendix section of this report. The Matlab scripts created for this project incorporate various Matlab functions from the LIGO Matlab Library. The directory is located on the test stand at:
/ligo/svncommom/SusSVN?sus/trunk/Commom/MatlabTools

One such function is vectfit4 [6], which takes a vector (transfer function data set) and fits it to a state-space model with a common zero and pole set (zpk). Another function called Autoquack (with related sub functions) prepares these zpk models and writes them to the appropriate component modules in the simulation schematic.

Two sets of recorded data were used to create two sets of filter arrays. The first set represents the frequency response of the ground to the sensors. This set of data was undamped and not stimulated. The second set of data is between the actuators of the system and the sensors. This set of data was recorded undamped with excitation. In order to build the noise filters, the frequency dependent transfer functions must be converted into equivalent frequency dependent filters that will simulate the system. The data we have used is from the Horizontal Access Module Internal Seismic Isolation System embedded in an .frd (frequency response data) Matlab file. The data set is in the Cartesian basis, converted from a local basis for the purpose of uniformity. The set was recorded overnight as a continuous time domain function and embedded into a discrete frequency domain matrix. The data consist of 6 actuators with 12 correlating sensors per actuator.

Scripts for the matrix filters were initially created to iterate a given number of times to obtain a more accurate representation. This method was changed to a converging method with an adjustable tolerance. This change produces a more accurate model of the data set while ensuring the program runs more efficiently. Once the matrices were created, they were tested and verified using a program named Foton. Foton (Filter Online Tool) that takes a zero pole gain (zpk) model and embeds a filter into a predefined Simulink filter block [9]. The tool reads and writes coefficient files for the online system to use in the specified block.

The scripts had to be revised several times in order to produce a better output. In addition to the adjustments mentioned above, the program was having difficulties accurately reproducing signals with very small frequency magnitudes. By adjusting the “weight” option in the vectfit4 function, the program was better able to simulate the response behavior.

6 REFERENCES

[1] Saulson, Peter R., Fundamentals of Interferometric Gravitational Wave Detectors. Singapore: World Scientific, 1994.

[2] Lantz, B., Schofield, R., O'Reilly, B., Clark, D. E., & DeBra, D., "Requirements for a Ground Rotation Sensor to Improve Advanced LIGO", Bulletin of the Seismological Society of America, vol. 99, no. 2B, pp. 980-989, May 2009

[3] Betweiser,B., *Controls with a Simulated Plant*. LIGO. https://dcc.ligo.org/DocDB/0061/G1100590/001/G1100590.pdf.

[4] Bork, R., and M. Aronsson, *AdvLigo CDS Real-time Code Generator (RCG) Application Developers Guide*. LIGO. https://dcc.ligo.org/public/0001/T080135/003/T080135-v3.pdf.

[5] Zhdanova, A., Real-time Simulation of a Suspended Cavity with the Advanced LIGO Digital Controls System. LIGO T1200462-x0-v2 (2012).

[6] Gustavsen, B. & Semlyen, A., "Rational approximation of frequency domain responses by Vector Fitting", IEEE Trans. Power Delivery, vol. 14, no. 3, pp. 1052-1061, July 1999.

[7] Gustavsen, B., "Improving the pole relocating properties of vector fitting", IEEE Trans. Power Delivery, vol. 21, no. 3, pp. 1587-1592, July 2006.

[8] Deschrijver, D., Mrozowski, M., Dhaene, T. & De Zutter, D., "Macromodeling of Multiport Systems Using a Fast Implementation of the Vector Fitting Method", IEEE Microwave and Wireless Components Letters, vol. 18, no. 6, pp. 383-385, June 2008.

[9] Sigg, D., *Online Digital Signal Processing*. LIGO. https://dcc.ligo.org/DocDB/0034/G020489/000/G020489-00.pdf.

7 ACKNOWLEDGEMENTS

First, I'd like to thank my mentor Joseph Betzweiser for all of his support and encouragement throughout the summer. Under his tutelage, I was able to learn and explore a large variety of topics. Thanks to Celine Ramet for providing us with additional data and helping work out some kinks in our system. Thanks to Ryan DeRosa, for helping me understand the HAM-ISI module and troubleshooting the simulation. Lastly, thank you to the National Science Foundation and the California Institute of Technology for the opportunity to participate in the Summer Undergraduate Research Fellowship program and the National Society of Hispanic Physicists for awarding me the Victor M. Blanco Undergraduate Summer Research Fellowship.

8 APPENDICES

CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK SCRIPT

```
% act_sensor_array.m
% CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\ACT_SENSOR\DATA\RAW_CART_DATA.mat'
);

% CONVERGENCE TOLERANCE
tol = 1e-7;

% MAXIMUM NO. ITERATIONS IN FIT
max_iter = 100;

% ACCEPTABLE INITIAL RMSERR VALUE
err = 1e-3;

% FREQUENCY GIVEN IN RAD/SEC
w = 1*1i*2*pi;

% ORDER OF APPROXIMATION
N = 9;

% VECTFIT OPTIONS
opts.relax=1;       %1 = Use vector fitting with relaxed non-triviality constraint
opts.stable=1;      %1 = Enforce stable poles
opts.asymp=3;       %3 = Include D and E in fitting
opts.skip_pole=0;   %1 = Skip pole identification
```

```
opts.skip_res=0;     %1 = Skip identification of residues (C,D,E)
opts.cmplx_ss=0;     %1 = Create complex state space model
opts.spy1=0;         %0 = No plotting for first stage of vector fitting
opts.spy2=0;         %1 = Create magnitude plot for fitting of f(s)
opts.logx=1;         %1 = Use logarithmic abscissa axis
opts.logy=1;         %1 = Use logarithmic ordinate axis
opts.errplot=0;      %1 = Include deviation in magnitude plot
opts.phaseplot=0;    %1 = Also produce plot of phase angle (in addition to magnitiude)
opts.legend=0;       %1 = include legends in plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOOP TO CREATE ZPK ARRAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter_array = ones([1,72]);
iter1 = 0;
to_quack = struct('name',{},'value',{},'label',{},'subblock', {});

for num_rows = 1:12                 % number of rows and columns
    for num_cols = 1:6
        iter1 = iter1 + 1;
        data = TF_C2C_Undamped_Symmetrized_frd(num_rows,num_cols);

        % EXTRACT RESPONSE(R) AND FREQUENCY SAMPLES(s)
        [R,s,ts] = frdata(data);
        s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
        Ns = length(s);

        % CREATE RESPONSE DATA MATRIX
        f = ones([1,Ns]);
        iter2 = 0;
        for k = 1:Ns
            iter2 = iter2 + 1;
            U = R(:,:,k);
            f([iter2,Ns]) = U;
        end

        % COMPLEX CONJUGATE PAIRS, LINEARLY SPACED
        bet = linspace(w(1),w(end),N/2);
        poles = [];
        for n = 1:length(bet)
            alf = -bet(n)*1e-2;
            poles = [poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
        end

        % ASSIGN WEIGHT FOR FREQUENCY POINTS
        %weight=ones(1,Ns);      % all frequency points are given equal weight
        %weight=1./(abs(f));     % strong inverse weight
        weight=1./(abs(f).^(2/3));
        %weight=1./sqrt(abs(f)); % weaker inverse weight

        % PERFORM VECTOR FIT
        disp('vector fitting...')
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

        % REPEAT ITERATION FOR IMPROVED ACCURACY
        rms = zeros(1);
        iter3 = 0;
        while rmserr > err && iter3 < max_iter
            iter3 = iter3 + 1;
%            disp(['   Iter ' num2str(iter)])
            [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
            rms(iter3,:) = rmserr;
            if iter3 < 2
```

```
                [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
                rms(iter3,:) = rmserr;
            elseif abs(rms(iter3)-rms(iter3-1)) > tol
                [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
                rms(iter3,:) = rmserr;
            else
                break;
            end
        end

        disp('Done.')
        A=full(SER.A);
        B=SER.B;
        C=SER.C;
        D=SER.D;
        E=SER.E;

        % Display Results
%         disp(['Converged in ' num2str(iter3) ' iterations.']);
%         disp('Resulting rms error convergence:');
%         rms
%         disp('Resulting zpk model:');

        zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

        % AUTOQUACK
        next_filt = struct('name','label','value','subblock');
        next_index = length(to_quack) + 1;
        next_filt.value = zpk_model;
        next_filt.name = ['HAM2_SIM_ACT_SENSOR_' num2str(num_rows) '_'
num2str(num_cols)];
        next_filt.label = ['ACT_SNSR_FILT'];
        next_filt.subblock = 0;
        to_quack(next_index) = next_filt;

        iter_array([iter1,72]) = iter3;
    end
end

disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);
```

CONVERT FREQUENCY FRDATA TO ZPK MODEL BY ITERATING SCRIPT

```
% frdatazpk_iter.m
% CONVERT FREQUENCY FRDATA TO ZPK MODEL BY ITERATING

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\H1_ISI_HAM2_TF_C2C_Raw_2013_05_08\
RAW_CART_DATA.mat');

% SPECIFIY FILE NAME AND CELL
data = TF_C2C_Undamped_Symmetrized_frd(8,3);
```

```
% INITIAL POLES FOR VECTOR FITTING
w = 1;

% ORDER OF APPROXIMATION
N = 10;

% PROGRAM ITERATIONS
Niter = 100;

% VECTFIT OPTIONS
opts.relax=1; %Use vector fitting with relaxed non-triviality constraint
opts.stable=1; %Enforce stable poles
opts.asymp=3; %Include both D, E in fitting
opts.skip_pole=0; %Do NOT skip pole identification
opts.skip_res=0; %Do NOT skip identification of residues (C,D,E)
opts.cmplx_ss=1; %Create complex state space model
opts.spy1=0; %No plotting for first stage of vector fitting
opts.spy2=1; %Create magnitude plot for fitting of f(s)
opts.logx=1; %Use logarithmic abscissa axis
opts.logy=1; %Use logarithmic ordinate axis
opts.errplot=1; %Include deviation in magnitude plot
opts.phaseplot=0; %Also produce plot of phase angle (in addition to magnitiude)
opts.legend=1; %Do include legends in plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PREPARE DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXTRACT RESPONSE(R) AND FREQUENCY POINRS(s)
[R,s] = frdata(data);
s = s*1i*2*pi;
Ns = length(s);

% CREATE RESPONSE MATRIX
f = ones([1,Ns]);
iter = 0;
for k=1:Ns
    iter = iter+1;
    U = R(:,:,k);
    f([iter,Ns]) = U;
end

% COMPLEX CONJUGATE PAIRS, LINEARLY SPACED
bet=logspace(w(1),w(end),N/2);
poles = [];
for n = 1:length(bet)
    alf = -bet(n)*1e-2;
    poles = [poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
end
poles=poles*2*pi;

% ASSIGN WEIGHT FOR FREQUENCY POINTS
weight=ones(1,Ns); %All frequency points are given equal weight

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PERFORM VECTOR FIT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('vector fitting...')

% REPEAT ITERATION FOR IMPROVED ACCURACY
for iter=1:Niter
  %Inculde legend in final plot
  if iter==Niter
```

```
      opts.legend=1;
  end
  disp(['   Iter ' num2str(iter)])
  [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
  rms(iter,1)=rmserr;
end
disp('Done.')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DISPLAY RESULTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Resulting state space model:')
A=full(SER.A)
B=SER.B
C=SER.C
D=SER.D
E=SER.E
rms
```

CONVERT FREQUENCY FRDATA TO ZPK MODEL BY CONVERGING SCRIPT

```
% frdatazpk_conv.m
% CONVERT FREQUENCY FRDATA TO ZPK MODEL BY CONVERGING

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GND_L4C.mat');

% SPECIFIY FILE NAME AND CELL
data = GND_L4C(7,2);

% CONVERGENCE TOLERANCE
tol = 1e-7;

% ACCEPTABLE INITIAL RMSERR VALUE
err = 1e-3;

% FREQUENCY GIVEN IN RAD/SEC
w = 1*1i*2*pi;

% ORDER OF APPROXIMATION
N = 9;

% VECTFIT OPTIONS
opts.relax=1;       %1 = Use vector fitting with relaxed non-triviality constraint
opts.stable=1;      %1 = Enforce stable poles
opts.asymp=3;       %3 = Include D and E in fitting
opts.skip_pole=0;   %1 = Skip pole identification
opts.skip_res=0;    %1 = Skip identification of residues (C,D,E)
opts.cmplx_ss=0;    %1 = Create complex state space model
opts.spy1=0;        %0 = No plotting for first stage of vector fitting
opts.spy2=1;        %1 = Create magnitude plot for fitting of f(s)
opts.logx=1;        %1 = Use logarithmic abscissa axis
opts.logy=1;        %1 = Use logarithmic ordinate axis
```

```
opts.errplot=0;      %1 = Include deviation in magnitude plot
opts.phaseplot=0;    %1 = Also produce plot of phase angle (in addition to magnitiude)
opts.legend=0;       %1 = include legends in plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PREPARE DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXTRACT RESPONSE(R) AND FREQUENCY SAMPLES(s)
[R,s,ts] = frdata(data);
s = s*1i*2*pi;               % frequency samples in j*w(rad/sec)
Ns = length(s);

% CREATE RESPONSE DATA MATRIX
f = ones([1,Ns]);
iter = 0;
for k = 1:Ns
    iter = iter+1;
    U = R(:,:,k);
    f([iter,Ns]) = U;
end

% COMPLEX CONJUGATE PAIRS, LINEARLY SPACED
bet = linspace(w(1),w(end),N/2);
poles = [];
for n = 1:length(bet)
    alf = -bet(n)*1e-2;
    poles = [poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
end

% ASSIGN WEIGHT FOR FREQUENCY POINTS
%weight=ones(1,Ns);      % all frequency points are given equal weight
%weight=1./(abs(f).^(2/3));     % strong inverse weight
weight=1./sqrt(abs(f)); % weaker inverse weight
%weight=1./(abs(f)); % weaker inverse weight
% weight=zeros(1,Ns);
% for k=1:Ns
%  weight(1,k)=1/sqrt(norm(f(:,k)));
% end
% ASSIGN WEIGHT FOR FREQUENCY POINTS
%weight=ones(1,Ns);      % all frequency points are given equal weight

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PERFORM VECTOR FIT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('vector fitting...')
[SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

% REPEAT ITERATION FOR IMPROVED ACCURACY
rms = zeros(1);
iter = 0;

while rmserr > err

    iter = iter + 1;
%    disp(['   Iter ' num2str(iter)])
    [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
    rms(iter,:) = rmserr;

    if iter<2
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    elseif abs(rms(iter)-rms(iter-1)) > tol
```

```
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    else
        break;
    end
end
disp('Done.')
A=full(SER.A);
B=SER.B;
C=SER.C;
D=SER.D;
E=SER.E;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DISPLAY RESULTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp(['Converged in ' num2str(iter) ' iterations.']);
disp('Resulting rms error convergence:');
rms
disp('Resulting zpk model:');
zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'))
```

CONVERT ASCII DATA FILE TO FRD MATRIX SCRIPT

```
% gnd_array.m
% CONVERT ASCII DATA FILE TO FRD MATRIX

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps1_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps1_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps1_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps2_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps2_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps2_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps3_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps3_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps3_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps4_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps4_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps4_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps5_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps5_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps5_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps6_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps6_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\CPS\cps6_3');

load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs7_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs7_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs7_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs8_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs8_2');
```

```
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs8_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs9_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs9_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs9_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs10_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs10_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs10_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs11_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs11_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs11_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs12_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs12_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GS13\gs12_3');

d1 =
frd(cps1_1(:,2)+cps1_1(:,3)*1i,cps1_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d2 =
frd(cps1_2(:,2)+cps1_2(:,3)*1i,cps1_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d3 =
frd(cps1_3(:,2)+cps1_3(:,3)*1i,cps1_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d4 =
frd(cps2_1(:,2)+cps2_1(:,3)*1i,cps2_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d5 =
frd(cps2_2(:,2)+cps2_2(:,3)*1i,cps2_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d6 =
frd(cps2_3(:,2)+cps2_3(:,3)*1i,cps2_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d7 =
frd(cps3_1(:,2)+cps3_1(:,3)*1i,cps3_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d8 =
frd(cps3_2(:,2)+cps3_2(:,3)*1i,cps3_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d9 =
frd(cps3_3(:,2)+cps3_3(:,3)*1i,cps3_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d10 =
frd(cps4_1(:,2)+cps4_1(:,3)*1i,cps4_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d11 =
frd(cps4_2(:,2)+cps4_2(:,3)*1i,cps4_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d12 =
frd(cps4_3(:,2)+cps4_3(:,3)*1i,cps4_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d13 =
frd(cps5_1(:,2)+cps5_1(:,3)*1i,cps5_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d14 =
frd(cps5_2(:,2)+cps5_2(:,3)*1i,cps5_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d15 =
frd(cps5_3(:,2)+cps5_3(:,3)*1i,cps5_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d16 =
frd(cps6_1(:,2)+cps6_1(:,3)*1i,cps6_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
```

```
d17 =
frd(cps6_2(:,2)+cps6_2(:,3)*1i,cps6_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d18 =
frd(cps6_3(:,2)+cps6_3(:,3)*1i,cps6_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');

d19 =
frd(gs7_1(:,2)+gs7_1(:,3)*1i,gs7_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d20 =
frd(gs7_2(:,2)+gs7_2(:,3)*1i,gs7_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d21 =
frd(gs7_3(:,2)+gs7_3(:,3)*1i,gs7_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d22 =
frd(gs8_1(:,2)+gs8_1(:,3)*1i,gs8_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d23 =
frd(gs8_2(:,2)+gs8_2(:,3)*1i,gs8_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d24 =
frd(gs8_3(:,2)+gs8_3(:,3)*1i,gs8_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d25 =
frd(gs9_1(:,2)+gs9_1(:,3)*1i,gs9_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d26 =
frd(gs9_2(:,2)+gs9_2(:,3)*1i,gs9_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d27 =
frd(gs9_3(:,2)+gs9_3(:,3)*1i,gs9_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d28 =
frd(gs10_1(:,2)+gs10_1(:,3)*1i,gs10_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d29 =
frd(gs10_2(:,2)+gs10_2(:,3)*1i,gs10_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d30 =
frd(gs10_3(:,2)+gs10_3(:,3)*1i,gs10_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d31 =
frd(gs11_1(:,2)+gs11_1(:,3)*1i,gs11_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d32 =
frd(gs11_2(:,2)+gs11_2(:,3)*1i,gs11_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d33 =
frd(gs11_3(:,2)+gs11_3(:,3)*1i,gs11_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d34 =
frd(gs12_1(:,2)+gs12_1(:,3)*1i,gs12_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d35 =
frd(gs12_2(:,2)+gs12_2(:,3)*1i,gs12_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d36 =
frd(gs12_3(:,2)+gs12_3(:,3)*1i,gs12_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');

% FREQUENCY RESPONSE DATA MATRIX
```

```
GND_L4C = [d1 d2 d3; d4 d5 d6; d7 d8 d9; d10 d11 d12; d13 d14 d15; d16 d17 d18; ...
    d19 d20 d21; d22 d23 d24; d25 d26 d27; d28 d29 d30; d31 d32 d33; d34 d35 d36]
```

CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK SCRIPT

```
% gnd_l4c_array.m
% CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_L4C\DATA\GND_L4C.mat');

% CONVERGENCE TOLERANCE
tol = 1e-7;

% MAXIMUM NO. ITERATIONS IN FIT
max_iter = 100;

% ACCEPTABLE INITIAL RMSERR VALUE
err = 1e-3;

% FREQUENCY GIVEN IN RAD/SEC
w = 1*1i*2*pi;

% ORDER OF APPROXIMATION
N = 9;

% VECTFIT OPTIONS
opts.relax=1;       %1 = Use vector fitting with relaxed non-triviality constraint
opts.stable=1;      %1 = Enforce stable poles
opts.asymp=3;       %3 = Include D and E in fitting
opts.skip_pole=0;   %1 = Skip pole identification
opts.skip_res=0;    %1 = Skip identification of residues (C,D,E)
opts.cmplx_ss=0;    %1 = Create complex state space model
opts.spy1=0;        %0 = No plotting for first stage of vector fitting
opts.spy2=0;        %1 = Create magnitude plot for fitting of f(s)
opts.logx=1;        %1 = Use logarithmic abscissa axis
opts.logy=1;        %1 = Use logarithmic ordinate axis
opts.errplot=0;     %1 = Include deviation in magnitude plot
opts.phaseplot=0;   %1 = Also produce plot of phase angle (in addition to magnitiude)
opts.legend=0;      %1 = include legends in plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOOP TO CREATE ZPK ARRAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter_array = ones([1,36]);
iter1 = 0;
to_quack = struct('name',{},'value',{},'label',{},'subblock', {});

for num_rows = 1:12                 % number of rows and columns
    for num_cols = 1:3
        iter1 = iter1 + 1;
        data = GND_L4C(num_rows,num_cols);
```

```
% EXTRACT RESPONSE(R) AND FREQUENCY SAMPLES(s)
[R,s,ts] = frdata(data);
s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
Ns = length(s);

% CREATE RESPONSE DATA MATRIX
f = ones([1,Ns]);
iter2 = 0;
for k = 1:Ns
    iter2 = iter2 + 1;
    U = R(:,:,k);
    f([iter2,Ns]) = U;
end

% COMPLEX CONJUGATE PAIRS, LINEARLY SPACED
bet = linspace(w(1),w(end),N/2);
poles = [];
for n = 1:length(bet)
    alf = -bet(n)*1e-2;
    poles = [poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
end

% ASSIGN WEIGHT FOR FREQUENCY POINTS
%weight=1./sqrt(abs(f)); % independent weaker inverse weight
weight=zeros(1,Ns);     % common weaker inverse weight
for k=1:Ns
 weight(1,k)=1/sqrt(norm(f(:,k)));
end

% PERFORM VECTOR FIT
disp('vector fitting...')
[SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

% REPEAT ITERATION FOR IMPROVED ACCURACY
rms = zeros(1);
iter3 = 0;
while rmserr > err && iter3 < max_iter
    iter3 = iter3 + 1;
%     disp(['   Iter ' num2str(iter3)])
    [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
    rms(iter3,:) = rmserr;
    if iter3 < 2
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter3,:) = rmserr;
    elseif abs(rms(iter3)-rms(iter3-1)) > tol
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter3,:) = rmserr;
    else
        break;
    end
end

disp('Done.')
A=full(SER.A);
B=SER.B;
C=SER.C;
D=SER.D;
E=SER.E;

% Display Results
%   disp(['Converged in ' num2str(iter3) ' iterations.']);
%   disp('Resulting rms error convergence:');
```

```
%          rms
%          disp('Resulting zpk model:');

        zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

        % AUTOQUACK
        next_filt = struct('name','label','value','subblock');
        next_index = length(to_quack) + 1;
        next_filt.value = zpk_model;
        next_filt.name = ['HAM2_SIM_GND_L4C_' num2str(num_rows) '_'
num2str(num_cols)];
        next_filt.label = ['GND_L4C_FILT'];
        next_filt.subblock = 0;
        to_quack(next_index) = next_filt;

        iter_array([iter1,36]) = iter3;
    end
end

disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);
```

CONVERT FREQUENCY ASCII TO ZPK MODEL BY CONVERGING SCRIPT

```
% gnd_pwrspctrm.m
% CONVERT FREQUENCY ASCII TO ZPK MODEL BY CONVERGING

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_PWRSPCTRM\DATA\GND_X');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_PWRSPCTRM\DATA\GND_Y');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\GND_PWRSPCTRM\DATA\GND_Z');

% SPECIFIY FILE NAME AND CELL
dataX = GND_X;
dataY = GND_Y;
dataZ = GND_Z;

% CONVERGENCE TOLERANCE
tol = 1e-7;

% ACCEPTABLE INITIAL RMSERR VALUE
err = 1e-3;

% FREQUENCY GIVEN IN RAD/SEC
w = 1*1i*2*pi;

% ORDER OF APPROXIMATION
N = 10;

% VECTFIT OPTIONS
opts.relax=1;       %1 = Use vector fitting with relaxed non-triviality constraint
opts.stable=1;      %1 = Enforce stable poles
opts.asymp=3;       %3 = Include D and E in fitting
```

```
opts.skip_pole=0;    %1 = Skip pole identification
opts.skip_res=0;     %1 = Skip identification of residues (C,D,E)
opts.cmplx_ss=0;     %1 = Create complex state space model
opts.spy1=0;         %0 = No plotting for first stage of vector fitting
opts.spy2=1;         %1 = Create magnitude plot for fitting of f(s)
opts.logx=1;         %1 = Use logarithmic abscissa axis
opts.logy=1;         %1 = Use logarithmic ordinate axis
opts.errplot=0;      %1 = Include deviation in magnitude plot
opts.phaseplot=0;    %1 = Also produce plot of phase angle (in addition to magnitiude)
opts.legend=0;       %1 = include legends in plots

% Complex conjugate pairs, logarithmically spaced :
bet=logspace(log10(w(1)),log10(w(end)),N/2);
poles=[];
for n=1:length(bet)
 alf=-bet(n)*1e-2;
 poles=[poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PREPARE DATA X
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXTRACT RESPONSE(f) AND FREQUENCY SAMPLES(s)
f = (dataX(:,2))';
s = dataX(:,1);
s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
Ns = length(s);

% ASSIGN WEIGHT FOR FREQUENCY POINTS
weight=ones(1,Ns);      % all frequency points are given equal weight

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PERFORM VECTOR FIT FOR X
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('vector fitting for x...')
[SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

% REPEAT ITERATION FOR IMPROVED ACCURACY
rms = zeros(1);
iter = 0;

while rmserr > err && iter < 100

    iter = iter + 1;
    disp(['   Iter ' num2str(iter)])
    [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
    rms(iter,:) = rmserr;

    if iter<2
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    elseif abs(rms(iter)-rms(iter-1)) > tol
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    else
        break;
    end
end
disp('Done.')
A=full(SER.A);
B=SER.B;
```

```
C=SER.C;
D=SER.D;
E=SER.E;
zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % AUTOQUACK FOR X
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
to_quack =
struct('name','HAM2_SIM_PWR_SPCTRM_X','value',zpk_model,'label','PWR_X_FILT','subblock
', 0);
disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);
%
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % %% PREPARE DATA Y
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % EXTRACT RESPONSE(f) AND FREQUENCY SAMPLES(s)
% % f = (dataY(:,2))';
% % s = dataY(:,1);
% % s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
% % Ns = length(s);
% %
% % % ASSIGN WEIGHT FOR FREQUENCY POINTS
% % % weight=zeros(1,Ns);
% % % for k=1:Ns
% % %  weight(1,k)=1/sqrt(norm(f(:,k)));
% % % end
% % % weight=zeros(1,Ns);
% % % for k=1:Ns
% % %  weight(1,k)=1/norm(f(:,k));
% % % end
% % weight=ones(1,Ns);      % all frequency points are given equal weight
% % %weight=1./(abs(f).^(2/3));     % strong inverse weight
% % %weight=1./sqrt(abs(f)); % weaker inverse weight
% % %weight=1./(abs(f)); % weaker inverse weight
% %
% %
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % %% PERFORM VECTOR FIT FOR Y
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % disp('vector fitting for y...')
% % [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
% %
% % % REPEAT ITERATION FOR IMPROVED ACCURACY
% % rms = zeros(1);
% % iter = 0;
% %
% % while rmserr > err && iter < 100
% %
% %     iter = iter + 1;
% %     disp(['   Iter ' num2str(iter)])
% %     [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
% %     rms(iter,:) = rmserr;
% %
% %     if iter<2
% %         [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
% %         rms(iter,:) = rmserr;
% %
% %     elseif abs(rms(iter)-rms(iter-1)) > tol
% %         [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
% %         rms(iter,:) = rmserr;
% %
```

```
% %     else
% %         break;
% %     end
% % end
% % disp('Done.')
% % A=full(SER.A);
% % B=SER.B;
% % C=SER.C;
% % D=SER.D;
% % E=SER.E;
% % zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% AUTOQUACK FOR Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
to_quack =
struct('name','HAM2_SIM_PWR_SPCTRM_Y','value',zpk_model,'label','PWR_Y_FILT','subblock
', 0);
disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREPARE DATA Z
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXTRACT RESPONSE(f) AND FREQUENCY SAMPLES(s)
f = (dataZ(:,2))';
s = dataZ(:,1);
s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
Ns = length(s);

% ASSIGN WEIGHT FOR FREQUENCY POINTS
for k=1:Ns
 weight(1,k)=1/norm(f(:,k));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PERFORM VECTOR FIT FOR Z
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('vector fitting for z...')
[SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

% REPEAT ITERATION FOR IMPROVED ACCURACY
rms = zeros(1);
iter = 0;

while rmserr > err && iter < 100

    iter = iter + 1;
    disp(['   Iter ' num2str(iter)])
    [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
    rms(iter,:) = rmserr;

    if iter<2
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    elseif abs(rms(iter)-rms(iter-1)) > tol
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
        rms(iter,:) = rmserr;

    else
        break;
    end
```

```
end
disp('Done.')
A=full(SER.A);
B=SER.B;
C=SER.C;
D=SER.D;
E=SER.E;
zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% AUTOQUACK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
to_quack =
struct('name','HAM2_SIM_PWR_SPCTRM_Z','value',zpk_model,'label','PWR_Z_FILT','subblock
', 0);
disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);
```

CREATE MATRIX INVERSE SCRIPT

```
% inv.matrices.m
% CREATE MATRIX INVERSE

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\INVERTED_MATRICES\DATA\GS132CART.t
xt');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\INVERTED_MATRICES\DATA\CPS2CART.tx
t');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\INVERTED_MATRICES\DATA\CART2ACT.tx
t');

SIM2GS13 = inv(GS132CART)
SIM2CPS = inv(CPS2CART)
ACT2SIM = inv(CART2ACT)

% sts_array.m
% CONVERT FREQUENCY ASCII TO FRD MATRIX

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps1_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps1_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps1_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps2_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps2_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps2_3');
```

```
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps3_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps3_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps3_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps4_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps4_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps4_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps5_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps5_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps5_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps6_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps6_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\cps6_3');

load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs7_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs7_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs7_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs8_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs8_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs8_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs9_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs9_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs9_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs10_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs10_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs10_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs11_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs11_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs11_3');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs12_1');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs12_2');
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\DATA\gs12_3');

d1 =
frd(cps1_1(:,2)+cps1_1(:,3)*1i,cps1_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d2 =
frd(cps1_2(:,2)+cps1_2(:,3)*1i,cps1_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d3 =
frd(cps1_3(:,2)+cps1_3(:,3)*1i,cps1_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d4 =
frd(cps2_1(:,2)+cps2_1(:,3)*1i,cps2_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d5 =
frd(cps2_2(:,2)+cps2_2(:,3)*1i,cps2_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d6 =
frd(cps2_3(:,2)+cps2_3(:,3)*1i,cps2_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d7 =
frd(cps3_1(:,2)+cps3_1(:,3)*1i,cps3_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d8 =
frd(cps3_2(:,2)+cps3_2(:,3)*1i,cps3_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d9 =
frd(cps3_3(:,2)+cps3_3(:,3)*1i,cps3_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d10 =
frd(cps4_1(:,2)+cps4_1(:,3)*1i,cps4_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
```

```
d11 =
frd(cps4_2(:,2)+cps4_2(:,3)*1i,cps4_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d12 =
frd(cps4_3(:,2)+cps4_3(:,3)*1i,cps4_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d13 =
frd(cps5_1(:,2)+cps5_1(:,3)*1i,cps5_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d14 =
frd(cps5_2(:,2)+cps5_2(:,3)*1i,cps5_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d15 =
frd(cps5_3(:,2)+cps5_3(:,3)*1i,cps5_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d16 =
frd(cps6_1(:,2)+cps6_1(:,3)*1i,cps6_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d17 =
frd(cps6_2(:,2)+cps6_2(:,3)*1i,cps6_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d18 =
frd(cps6_3(:,2)+cps6_3(:,3)*1i,cps6_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');

d19 =
frd(gs7_1(:,2)+gs7_1(:,3)*1i,gs7_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d20 =
frd(gs7_2(:,2)+gs7_2(:,3)*1i,gs7_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d21 =
frd(gs7_3(:,2)+gs7_3(:,3)*1i,gs7_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d22 =
frd(gs8_1(:,2)+gs8_1(:,3)*1i,gs8_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d23 =
frd(gs8_2(:,2)+gs8_2(:,3)*1i,gs8_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d24 =
frd(gs8_3(:,2)+gs8_3(:,3)*1i,gs8_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d25 =
frd(gs9_1(:,2)+gs9_1(:,3)*1i,gs9_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d26 =
frd(gs9_2(:,2)+gs9_2(:,3)*1i,gs9_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d27 =
frd(gs9_3(:,2)+gs9_3(:,3)*1i,gs9_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit','H
z');
d28 =
frd(gs10_1(:,2)+gs10_1(:,3)*1i,gs10_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d29 =
frd(gs10_2(:,2)+gs10_2(:,3)*1i,gs10_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d30 =
frd(gs10_3(:,2)+gs10_3(:,3)*1i,gs10_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
```

```
d31 =
frd(gs11_1(:,2)+gs11_1(:,3)*1i,gs11_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d32 =
frd(gs11_2(:,2)+gs11_2(:,3)*1i,gs11_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d33 =
frd(gs11_3(:,2)+gs11_3(:,3)*1i,gs11_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d34 =
frd(gs12_1(:,2)+gs12_1(:,3)*1i,gs12_1(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d35 =
frd(gs12_2(:,2)+gs12_2(:,3)*1i,gs12_2(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');
d36 =
frd(gs12_3(:,2)+gs12_3(:,3)*1i,gs12_3(:,1),1/4096,'Name','HAM_ISI_GND','FrequencyUnit'
,'Hz');

STS_SENSOR = [d1 d2 d3; d4 d5 d6; d7 d8 d9; d10 d11 d12; d13 d14 d15; d16 d17 d18; ...
    d19 d20 d21; d22 d23 d24; d25 d26 d27; d28 d29 d30; d31 d32 d33; d34 d35 d36]
```

CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK SCRIPT

```
% sts_sensor.m
% CONVERT FREQUENCY FRDATA TO ZPK MODEL AND APPLY AUTOQUACK

% INITIALIZE MATLAB
close all;
clc;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DASHBOARD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD FREQUENCY DATA
load('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\STS_SENSOR\STS_SENSOR.mat');

% CONVERGENCE TOLERANCE
tol = 1e-9;

% MAXIMUM NO. ITERATIONS IN FIT
max_iter = 100;

% ACCEPTABLE INITIAL RMSERR VALUE
err = 1e-5;

% FREQUENCY GIVEN IN RAD/SEC
w = 1*1i*2*pi;

% ORDER OF APPROXIMATION
N = 9;

% VECTFIT OPTIONS
opts.relax=1;       %1 = Use vector fitting with relaxed non-triviality constraint
opts.stable=1;      %1 = Enforce stable poles
opts.asymp=3;       %3 = Include D and E in fitting
opts.skip_pole=0;   %1 = Skip pole identification
opts.skip_res=0;    %1 = Skip identification of residues (C,D,E)
opts.cmplx_ss=0;    %1 = Create complex state space model
opts.spy1=0;        %0 = No plotting for first stage of vector fitting
```

```
opts.spy2=1;         %1 = Create magnitude plot for fitting of f(s)
opts.logx=1;         %1 = Use logarithmic abscissa axis
opts.logy=1;         %1 = Use logarithmic ordinate axis
opts.errplot=0;      %1 = Include deviation in magnitude plot
opts.phaseplot=0;    %1 = Also produce plot of phase angle (in addition to magnitiude)
opts.legend=0;       %1 = include legends in plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOOP TO CREATE ZPK ARRAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter_array = ones([1,36]);
iter1 = 0;
to_quack = struct('name',{},'value',{},'label',{},'subblock', {});

for num_rows = 1:12                 % number of rows and columns
    for num_cols = 1:3
        iter1 = iter1 + 1;
        data = STS_SENSOR(num_rows,num_cols);

        % EXTRACT RESPONSE(R) AND FREQUENCY SAMPLES(s)
        [R,s,ts] = frdata(data);
        s = s*1i*2*pi;              % frequency samples in j*w(rad/sec)
        Ns = length(s);

        % CREATE RESPONSE DATA MATRIX
        f = ones([1,Ns]);
        iter2 = 0;
        for k = 1:Ns
            iter2 = iter2 + 1;
            U = R(:,:,k);
            f([iter2,Ns]) = U;
        end

        % COMPLEX CONJUGATE PAIRS, LINEARLY SPACED
        bet = linspace(w(1),w(end),N/2);
        poles = [];
        for n = 1:length(bet)
            alf = -bet(n)*1e-2;
            poles = [poles (alf-1i*bet(n)) (alf+1i*bet(n)) ];
        end

        % ASSIGN WEIGHT FOR FREQUENCY POINTS
%         for k=1:Ns
%          weight(1,k)=1/sqrt(norm(f(:,k)));
%         end
%         weight=1./(abs(f).^(2/3));     % strong inverse weight
         weight=1./sqrt(abs(f)); % weaker inverse weight

        % PERFORM VECTOR FIT
        disp('vector fitting...')
        [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);

        % REPEAT ITERATION FOR IMPROVED ACCURACY
        rms = zeros(1);
        iter3 = 0;
        while rmserr > err && iter3 < max_iter
            iter3 = iter3 + 1;
            disp(['   Iter ' num2str(iter3)])
            [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
            rms(iter3,:) = rmserr;
            if iter3 < 2
                [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
                rms(iter3,:) = rmserr;
```

```
            elseif abs(rms(iter3)-rms(iter3-1)) > tol
                [SER,poles,rmserr,fit]=vectfit4(f,s,poles,weight,opts);
                rms(iter3,:) = rmserr;
            else
                break;
            end
        end

        disp('Done.')
        A=full(SER.A);
        B=SER.B;
        C=SER.C;
        D=SER.D;
        E=SER.E;

        % Display Results
%         disp(['Converged in ' num2str(iter3) ' iterations.']);
%         disp('Resulting rms error convergence:');
%         rms
%         disp('Resulting zpk model:');

        zpk_model = zpk(c2d(ss(A,B,C,D),1/4096,'tustin'));

        % AUTOQUACK
        next_filt = struct('name','label','value','subblock');
        next_index = length(to_quack) + 1;
        next_filt.value = zpk_model;
        next_filt.name = ['HAM2_SIM_GND_L4C_' num2str(num_rows) '_'
num2str(num_cols)];
        next_filt.label = ['GND_L4C_FILT'];
        next_filt.subblock = 0;
        to_quack(next_index) = next_filt;

        iter_array([iter1,36]) = iter3;
    end
end

disp(' ');
autoquack('C:\Users\Fernie\Documents\MATLAB\LIGO\HAM_ISI\X2ISIHAM2SIM.txt', to_quack);
```