Dynamic Signal Analyzer for dSPACE

K. A. Lilienkamp, and D. L. Trumper
Precision Motion Control Lab, MIT, Rm 35-030, 77 Mass. Ave, Cambridge, MA 02139

We have developed MATLAB software and a Simulink subsystem block which work in tandem to extract the transfer function (amplitude and phase) of a system using 'swept sine' excitation. The resulting dynamic signal analyzer provides a convenient tool for obtaining empirical Bode plots of system and controller dynamics within the dSPACE environment. Both the derived (Fourier) and raw (time domain) data are displayed at each frequency tested. The bandwidth possible is linearly related to the sampling rate limits set by both the dSPACE board used and the complexity of the model. With the ds1102 and a simple model, at a sampling rate of 10 kHz, the measurement bandwidth is about 1 kHz.

1. Motivation

Characterizing both time and frequency domain response is a necessary step in understanding system dynamics. In the field of controls, designers use this information both for system identification and for validation of controller behavior. The dSPACE environment provides useful graphical tools for recording and displaying discrete-time data (e.g., the step response of a system). We saw a need for similar tools that could provide empirical Bode plots of the frequency response without interfacing to external hardware. One particular benefit of a signal analyzer that runs within a Simulink model over an external DSA is the ease with which the various signals can be used without the necessity of transforming them into analog representations.

This project was the first author's Bachelor's thesis to develop a dynamic signal analyzer (DSA) for the ds1102.[1] We have used this DSA on recent projects at the Precision Motion Control Lab at MIT. The source code for the signal analyzer is publicly available at our website, with the expectation others will find it useful as well.[2]

2. Theory and Implementation

Our dSPACE-based DSA uses 'swept sine excitation' to determine the gain and phase of a system at a set of desired frequencies. Below is a brief description of the theory behind this method of system identification. The goal is to provide enough information to understand the algorithms outlined in this section.[3] A MATLAB function implementing this swept sine method for determining the empirical system transfer function is one fundamental piece of our dSPACE dynamic signal analyzer. The second component of the DSA is a subsystem block, as described further below.

The basic method necessary at each frequency of interest is to excite a system with a sine wave of known phase and amplitude and to observe the phase and amplitude of the response. If the system response is relatively clean, it is not difficult to determine these quantities. (In figure 1, for instance, the response at right has a gain of 2x and a phase shift of 90 degrees, relative to the input at left.)

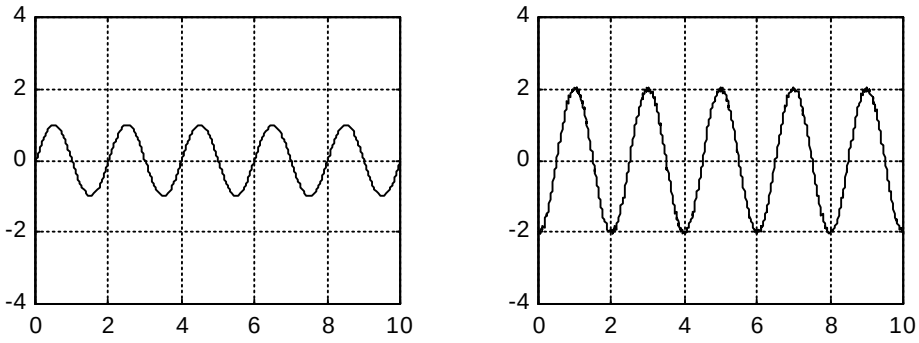

Figure 1. Sine excitation (left) and a relatively clean response (right).

In general, the system response may be much noisier and may contain underlying noise components that do not permit such an easy interpretation of the gain and phase (see figure 2). In figure 1, simply evaluating the magnitude and time delay of the peaks and troughs in the signal and comparing these with those of the known excitation signal can provide reasonable estimates of gain and phase. This is clearly not a workable approach for the response shown in figure 2, however.

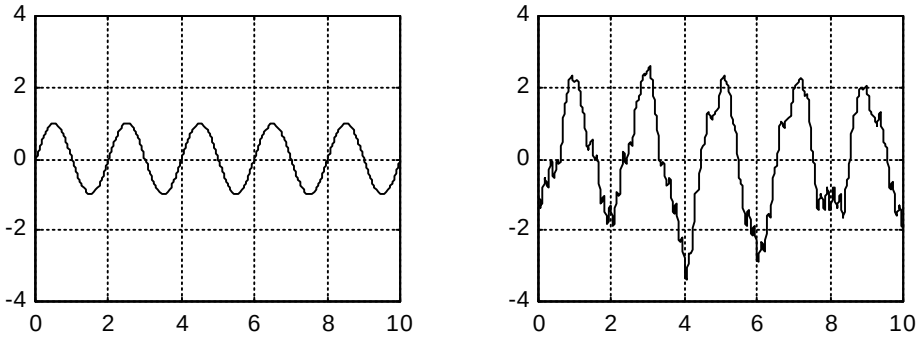

Figure 2. Sine excitation (left) and noisy response with harmonics (right).

To develop a more general methodology, we can exploit two basic ideas. The first is that a signal can be represented as a sum of sine wave components, each with a specific amplitude and phase. These are the Fourier components of the signal. Second, we can use the orthogonality of each harmonic with every other. Each sine wave is orthogonal to every sine wave at a different frequency, and sine and cosine waves (i.e., waves separated from one another by exactly 90 degrees) are also orthogonal to one another.

Two vectors that are orthogonal are also uncorrelated. The correlation between two vectors can be found by calculating the average value (the expected value) of their product over all time. The dynamic signal analyzer thus takes the product of a pure sine wave at the excitation frequency with the output signal of the system. If the data were infinite length, the components in the analyzed signal due to a harmonic at any other frequency contributes nothing to this product, because these two *different* harmonics are uncorrelated. In a finite-length signal there will be a residual contribution, but we do not consider this further here. The average of this product, *over an integral number of sine waves*, is linearly proportional to the amplitude of the sine wave component of the output signal at the desired frequency. Following the development and notation in [3], pp.484-495, the amplitude of the cosine component of this harmonic is given by:

$$B_c = \frac{2}{N}\sum_{k=0}^{N-1} y(kT)\cos(\omega kT)$$

Similarly, the amplitude of the sine component of the signal is

$$B_s = \frac{2}{N}\sum_{k=0}^{N-1} y(kT)\sin(\omega kT)$$

where k is the sampling index, N is the number of samples, and T is the sampling interval. Using trigonometic identities, we can then derive for each of the two DSA input signals both the total amplitude of harmonic component at this frequency:

$$B = \sqrt{(B_c^2 + B_s^2)}$$

and the phase shift:

$$\phi = \operatorname{atan}\frac{B_c}{B_s}$$

Finally, the gain of the transfer function from channel 1 to channel 2 is simply

$$\left|G(e^{j\omega T})\right| = \frac{B_2}{B_1}$$

and the phase shift is simply the difference in the individual phases of the two signals:

$$\angle G(e^{j\omega T}) = \phi_2 - \phi_1$$

A flowchart of the MATLAB algorithm is shown in figures 3-A and 3-B (on the next two pages) that describes how the data are collected and processed.

The second component of the DSA, in addition to the MATLAB function, is a Simulink subsystem block. The block has one output, which provides the exciting sine wave signal. The DSA compares signals at its two inputs to compute the transfer function from channel 1 to channel 2. The MATLAB function automatically modifies the frequency and amplitude of the sine wave output at each frequency that is tested (see appendix for more details). Figure 3 shows the DSA subsystem block (at left) connected to analyze a simple zero-pole subsystem. Note that it is frequently convenient but not necessary that the output signal ('SineOut') be connected to the input at channel 1. The two channels can receive data from any two nodes of interest within the system.

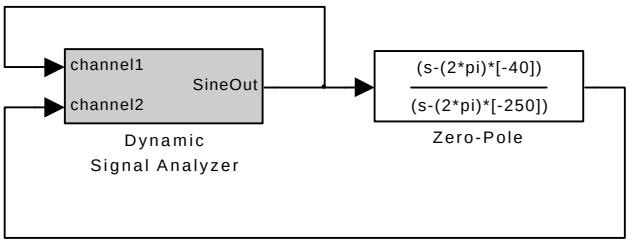

Figure 3. DSA Simulink block exciting a zero-pole subsystem.

Flow Chart for the MATLAB DSA Function:

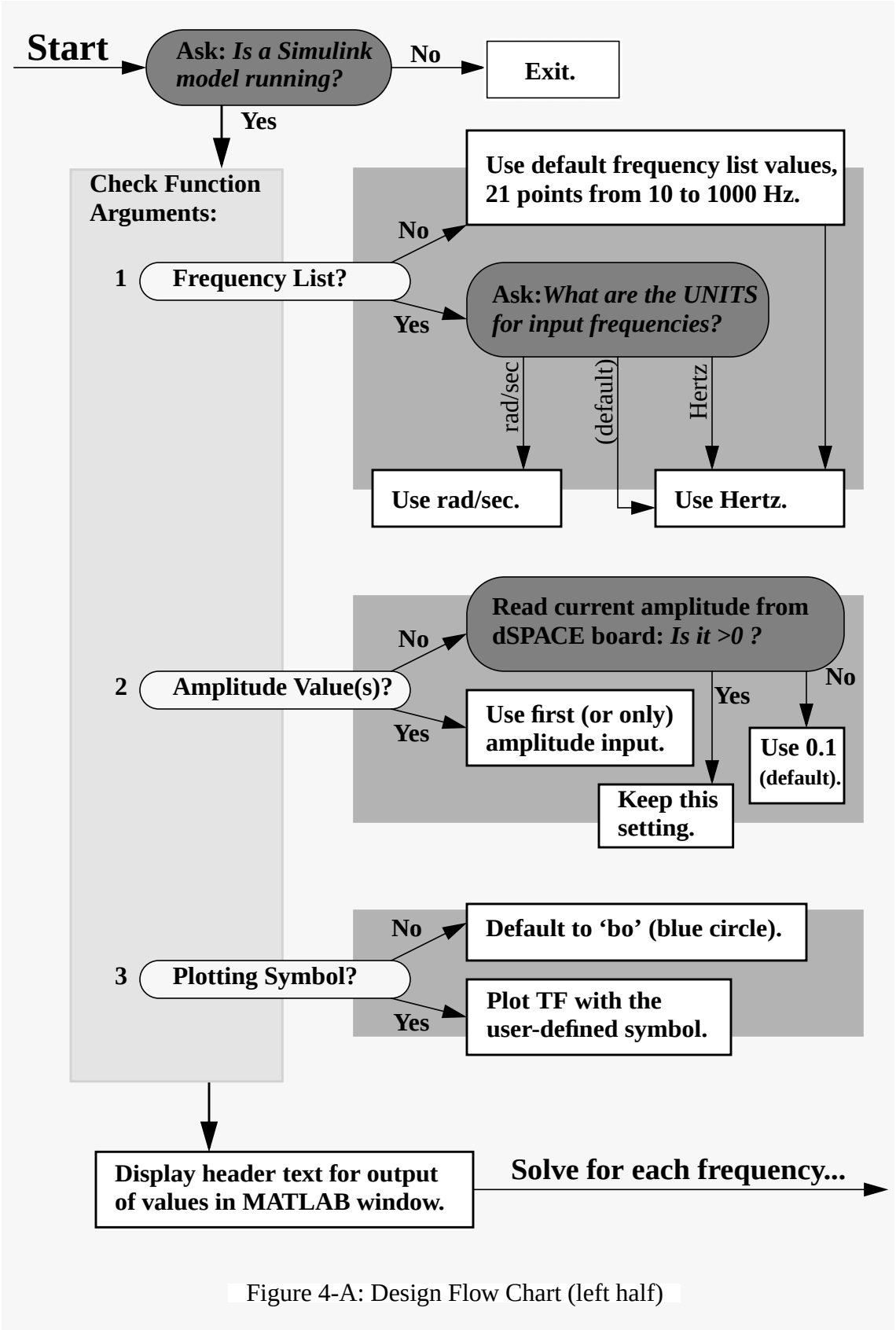

Figure 4-A: Design Flow Chart (left half)

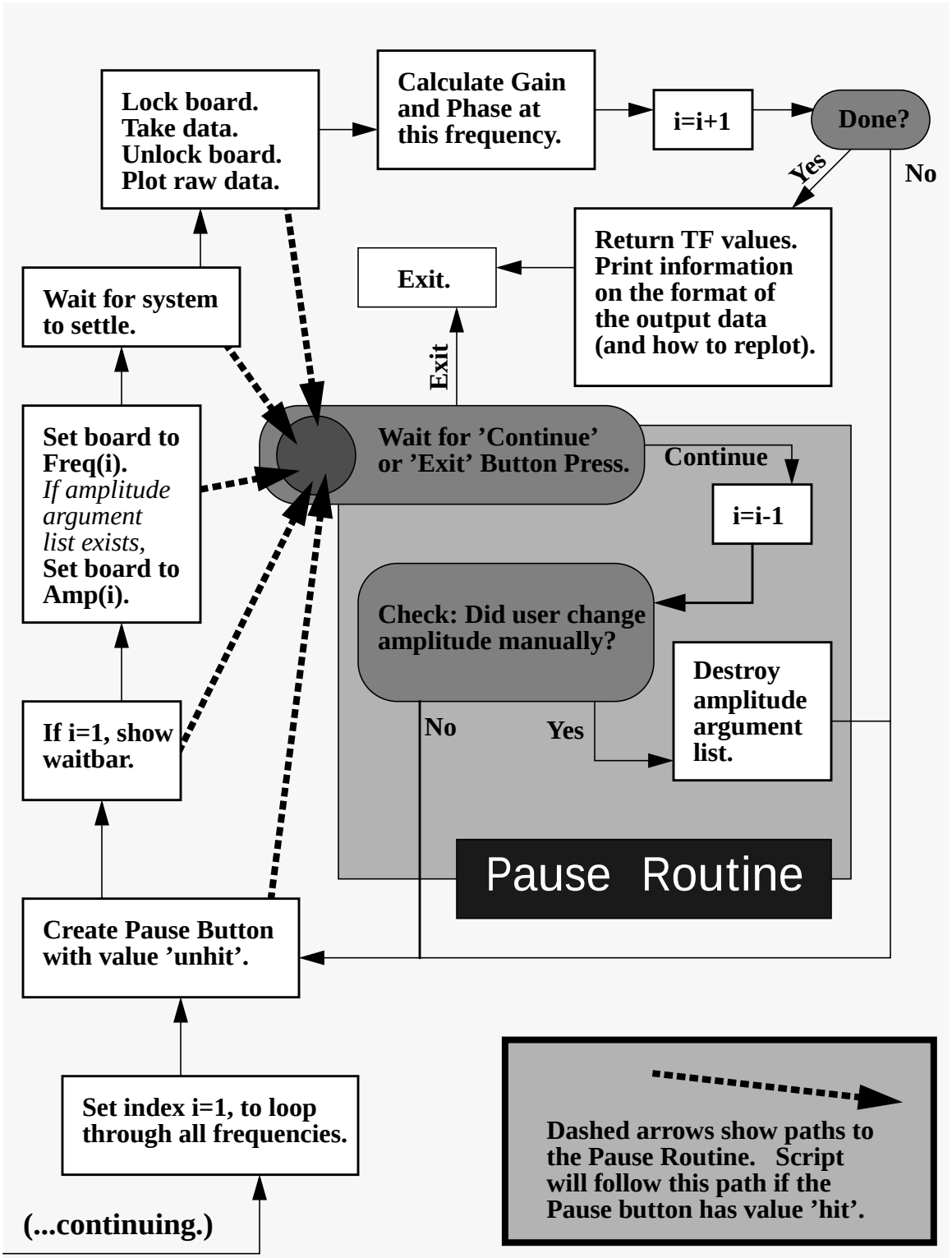

Figure 4-B: Design Flow Chart (right half)

3. Operation of the DSA

The MATLAB function takes up to three input arguments. The first is a vector of frequencies. The user is asked whether the frequencies are given in Hertz of radians per second at the start of the run. The default is to use Hertz.

The second argument is a vector of amplitudes corresponding to the amplitude of the output sine wave from the DSA subsystem block at each frequency. If the value input is a scalar or is shorter than the list of frequencies, the last (or only) amplitude in the set is used for all remaining frequencies to be tested. If the system dynamics vary significantly over the frequency range being tested, the user may wish to define different amplitudes for different frequencies. For example, an output sine wave amplitude which is so large it nearly saturates the input channels at a low frequency may turn out to be much too weak to adequately excite the same system at higher frequencies. In such a case, the resulting sine wave components may be too small to extract a meaningful signal from the DSA input channels, requiring larger amplitudes of excitation at higher frequencies.

The third argument to the function is an optional plotting symbol, provided in the standard syntax used within MATLAB. (The string 'bo-' indicates blue circles connected with a solid, blue line, for instance. This is the default plotting option used.) If the DSA function is called multiple times without clearing the MATLAB figure between runs, the use of distinct plotting symbols will distinguish the various transfer functions from one another.

As each frequency is tested, the DSA updates a cumulative, derived Bode plot of the system. A separate figure displays samples of the data collected. The user should examine these raw data streams to insure that neither of the inputs is saturated nor buried in noise. An example of these two output windows is shown in Figure 5.

If either signal seems inadequate, or if there is any other reason to halt the program, there is a pause button in one of the figures. After interrupting the program, the user can modify the amplitude of the DSA output sine wave, using either Cockpit or the ContolDesk environment. If the same run is then continued, the last tested frequency will be rerun, using any modified value for the sine amplitude. Alternatively, the user can then select a complete break of the program. Stopping the run by using these buttons is preferable to breaking within the MATLAB window (using ctrl-c). Otherwise, the program may stop after a 'LockProgram' call to mtrace, requiring the user to enter 'UnlockProgram' before performing further runs.

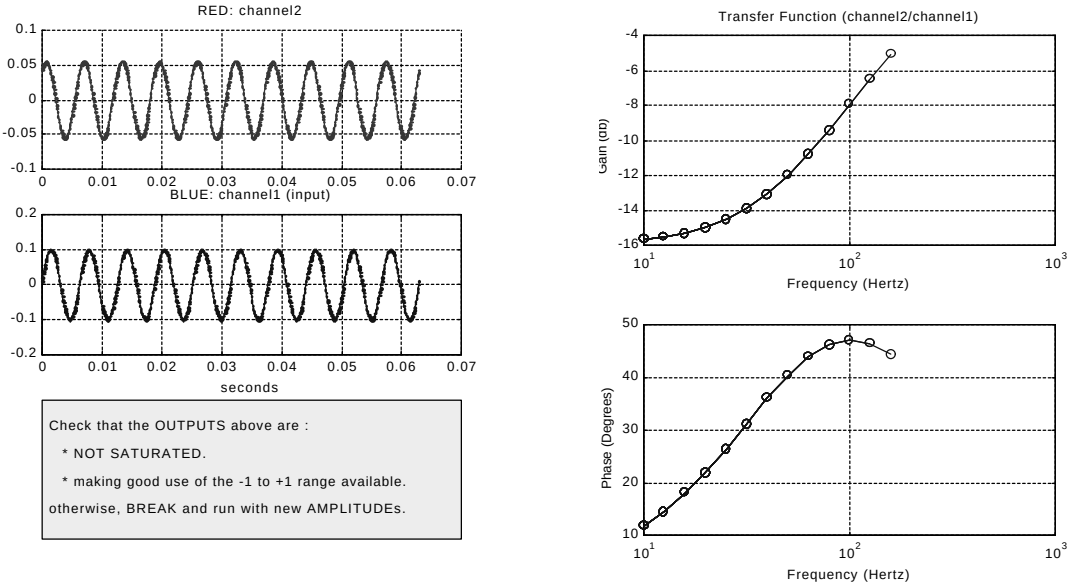

Figure 5. Samples of the input data (left) and the derived Bode plot (right).

The MATLAB function provides an output matrix of three column vectors: the list of frequencies at which the system was tested (in the units defined by the user), the gain at each frequency (as a ratio, not in db), and the phase (in degrees). All the exact details of the inputs and outputs of the function are given in the 'help' information at the head of the DSA file.

One typical use of the dynamic signal analyzer is the measurement of loop transmission of a controlled system. Below, figure 6 shows a closed-loop system with an unspecified controller within the Simulink model interfacing to some external system (the plant) via D/A and A/D channels of the dSPACE board. Additional connections necessary for measurement of the loop transmission are shown in figure 7.

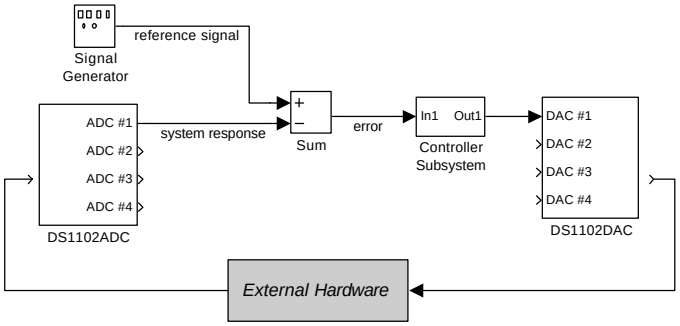

Figure 6. Closed-loop system

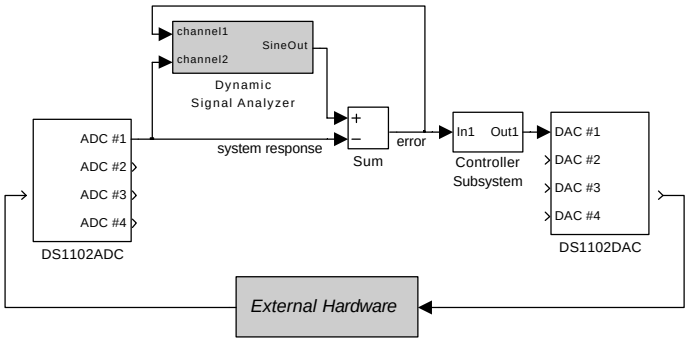

Figure 7. Measurement of the loop-transmission of a controlled system.

4. System Requirements and Limitations

The dynamic signal analyzer was developed to run on the ds1102, but it can be adapted for other dSPACE boards as well. The primary modifications necessary for it to be used with other dSPACE boards involve the syntax within the MATLAB function. Specifically, all calls to 'mtrace' functions within the function use a different syntax on other boards and with different versions of mtrace. For example, the expression below, originally used for the ds1102,

mtrc31('SelectBoard','ds1102')

would need to be replaced with

mtrc1103('SelectBoard','ds1103')

to operate with the ds1103. Other versions of the dSPACE board may require additional modifications in the syntax of the code, but this is the main incompatibility we have found to date.

The bandwidth over which the Bode plot can be measured is limited by the sampling rate of the Simulink model which is being tested. The Nyquist criterion requires at least two sample points per cycle to avoid aliasing. Because the sine excitation output by the DSA subsystem block is discrete, there must actually be enough sample points per cycle to create a relatively 'smooth' sine wave to excite the system being tested. We recommend operating with at least ten discrete date points per cycle. For a Simulink "fixed-step" size of 0.1 microseconds, for instance, this maximum frequency would be 1.0 kHz

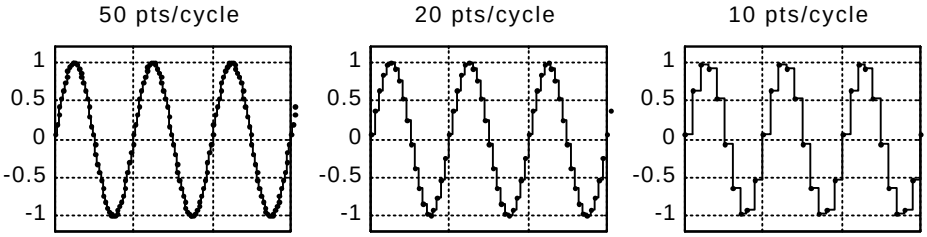

Figure 8. Discrete sine wave approximations.

For frequencies below about 10 Hz, it may also be necessary to adapt the code to take multiple trace streams of data to obtain several complete cycles, since at most 10,000 points are collected in a single trace capture. With a step size of 0.1 microseconds, this is just one second of data. One solution is to store multiple traces of data and to average them. We have added additional lines of code to perform this type of averaging for the analysis of a vibration isolation table. For this application, the code was modified to analyze frequencies as low as 0.1 Hz.

As a final comment on system requirements, note that the system to be tested can (and will, generally) involve a combination of a Simulink model and external hardware. You can drag a DSA subsystem block into a new system of interest, just as other Simulink and dSPACE blocks can be dragged from libraries into a model. The DSA block operates as any other Simulink subsystem, with one exception; you cannot use multiple DSA blocks within the same model. The MATLAB function that runs the signal analysis expects the block to have the specific (and unique) name "Dynamic Signal Analyzer". A duplicate would have a name such as "Dynamic Signal Analyzer2". This block would not be identified by the script file as a valid DSA block.

5. Experimental Results

Below are data comparing the transfer function for a simple, low-pass RC circuit obtained from the dSPACE DSA with that from an HP 35662 dynamic signal analyzer manufactured by Hewlett-Packard. The circuit consists of a 73 kOhm resistor and a 0.022 microFarad capacitor, resulting in a breakpoint of 99 Hz as shown below. The solid lines show the response found by the HP machine, which is very close to the expected, theoretical output. The points represent the functions derived by the dSPACE DSA.

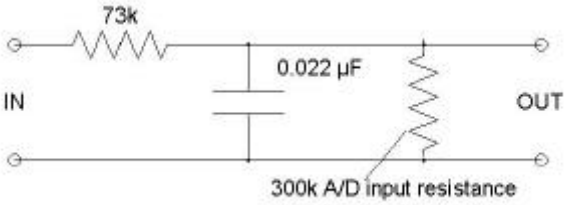

Figure 9. Low-Pass RC circuit, showing on-board 300k resistance to ground.

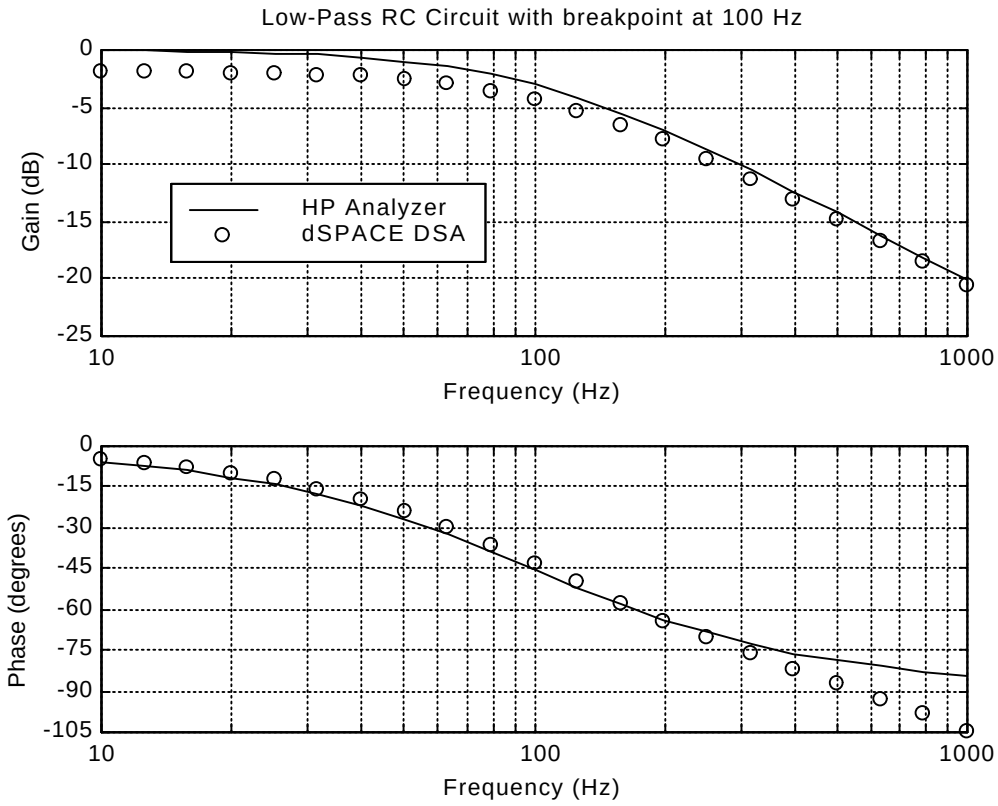

Figure 10. Low-Pass RC Circuit Response

There are two primary discrepancies between the expected transfer functions and the one obtained by the dSPACE DSA. First, the gain values at low frequencies are approximately 2 db lower than the predicted values. This occurs because the RC circuit resistance is of the same order of magnitude as the resistance of dSPACE A/D channel to ground, which creates a voltage divider. The RC resistor is 73 kOhm and the effective resistance to ground is about 300 kOhm for the A/D channels on the 1102. The resulting low-frequency output voltage measured by the board is therefore can be calculated as shown:

$$V_{meas} = (R_{board}/R_{tot}) * V_{RC} = (300/(300+73)) * V_{RC} = \sim 0.8 * V_{RC}$$

This means the board measures only about 80% of the actual out from the RC circuit, which corresponds to the 2db difference between the expected and empirical values.

The second and more significant error in the dSPACE DSA output occurs in the calculation of the phase at higher frequencies. The discrete sampling of the data results in a one-half sample delay in the phase. With a sampling period of 1.0e-4 seconds, this amounts to a delay of 0.5e-4 seconds. For a frequency of 1 kHz, the resulting phase lag is about 18 degrees:

$$\text{Phase Lag from Half Sample Delay} = 360^{o} * [0.5e\text{-}4\ (\text{sec}) * 1e3\ (\text{Hz})] = 18^{o}$$

This is simply a general result of discrete sampling. If desired the user can alter the code to correct for this phase lag at each discrete frequency tested by calculating the predicted phase lag and adding this value to the phase derived directly from the existing algorithms.

We have tested the dynamic signal analyzer with various systems, and we have used it to analyze system dynamics in several recent projects at the Precision Motion Control Lab. One recent example is the PhD dissertation of Ming-chih Weng.[4] Here, the code was modified to be compatible with the ds2201, and

the transfer function obtained actually consisted of matrices of gain and phase, relating several sensors and actuators.

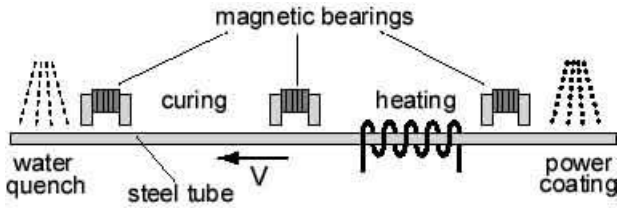

Figure 11. Non-contact coating of a long, slender beam.

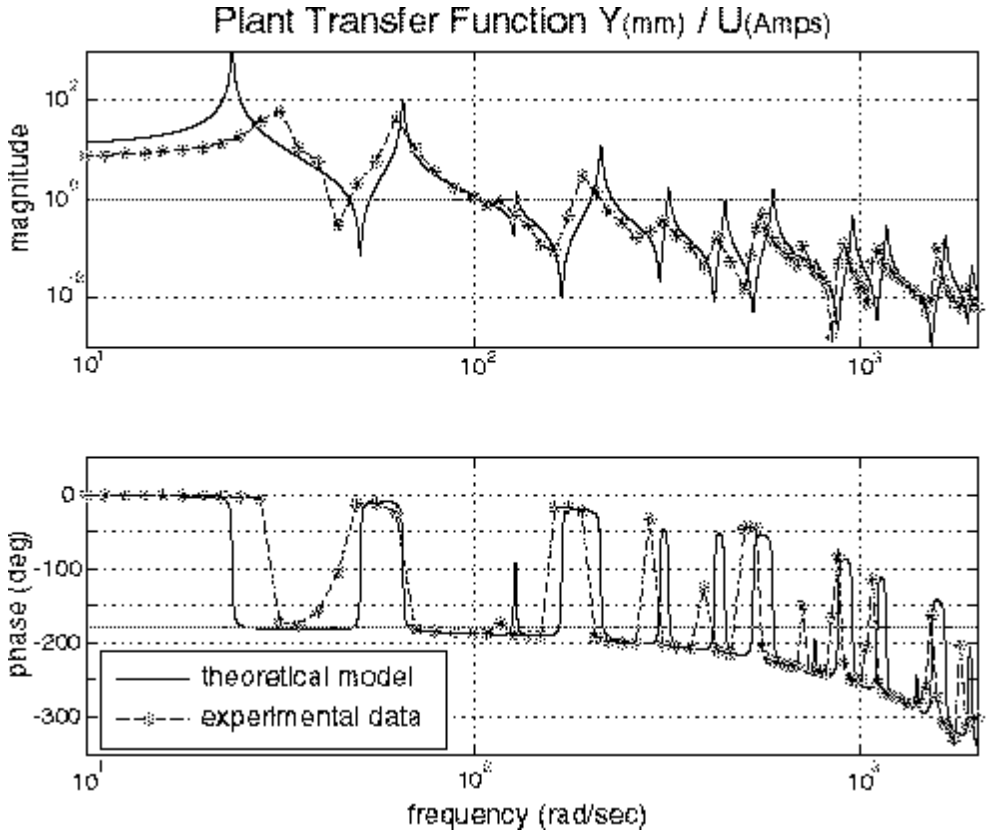

Figure 12. Bode plots obtained with the dSPACE DSA (suspended beam dynamics). Solid line is from theory; dashed line is measured data from the DSA.

Figure 11 shows a schematic of the project, which studies magnetic levitation of flexible beams during painting and curing processes. The original inception applied specifically to the manufacture of metal broomstick handles. Touching the broomstick before the paint has set results in wasted material at those locations, which then needs to be cut away and discarded. Stable levitation of the rod eliminates this waste, but the dynamics of such a long, slender structure present a complex control problem. A typical transfer function obtained with this version the dSPACE DSA is presented in figure 12. Our dSPACE DSA provides a useful and convenient means of evaluating the system's dynamics.

6. Adaptability and Further Development

As previously mentioned, both the Simulink subsystem block and the MATLAB function to run the dSPACE DSA are publicly available at the PMC website at MIT. While we believe this is a useful and adaptable tool, the Precision Motion Control Lab does not guarantee its performance nor take responsibility for the results from its use.

Some of the possibilities for furthering the potential of dynamic signal analyzer are listed earlier. These include necessary 'mtrace' modifications for compatibility with different dSPACE boards and versions of mtrace. A wide variety of refinements are also possible. A short list includes the following:

- adjusting for the effects of the half sample phase delay
- averaging multiple traces of data to obtain better results at low frequencies
- automatically detecting saturation at the inputs and then scaling the output amplitude of the DSA to avoid this
- searching for local maximums in the gain through automatic, step-wise selection of intermediate frequencies near the peaks. (M. C. Weng implemented such an algorithm to aide in analyzing the dynamics of the levitating broomstick, shown in figure 11.)
- using more sophisticated estimation algorithms

Notes

[1] K. A. Lilienkamp. "A Simulink-Driven Dynamic Signal Analyzer," Bachelor's Thesis, MIT Dept. of Mechanical Engineering, 1999. (http://web.mit.edu/gonzo/Public/thesis1.pdf)

[2] Both the files which comprise our dSPACE DSA are available at the following website:
http://web.mit.edu/pmc/www/Links/download/
Guidelines and instruction for use can also be found at this site. The Precision Motion Control Lab at MIT is distributing this software for free. We make no claims to the accuracy or reliability of this tool, and we are not liable for any consequences resulting from its use.

[3] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems [3rd edition]*. (Reading, Addison-Wesley, 1990) Pages 484-495 contain an excellent discussion of non-parametric system identification methods, including swept sine response.

[4] M. C. Weng. "Magnetic and Electrostatic Suspension Control of Flexible Structures for Non-contact Processing of Fibers, Beams, Webs, and Plates," Ph.D. Thesis, MIT Dept. of Mechanical Engineering, 2000. (Project details available at: http://web.mit.edu/pmc/www/Newprojects/Broomlev/broomlev.html)

Appendix: Partial listing of the source code.

The steps below implement the method of swept sine extraction of the non-parametric transfer function that is described in section 2 of this paper. Some of the relevant MATLAB code is listed with each.

1. Identify the sample step size of the model. This is the time elapsed between data points.

```
samp_per=mlib('GetSimAddr','Task Info/Timer Task 1/sampleTime');
dt=mlib('Readd',samp_per);
```

2. Select the output and two input channels as trace variables for data collection.

```
y_addr=mtrc31('GetAddr','rti B[Model Root/Dynamic Signal Analyzer/channel1]',...
    'rti B[Model Root/Dynamic Signal Analyzer/channel2]',...
    'rti B[Model Root/Dynamic Signal Analyzer/Swept Sine]');
mtrc31('TraceVars',y_addr);
```

3. Determine number of points needed to create an integral number of cycles.

```
w=w_use(i);     % frequency for this data set
ncyc=floor(w*dt*N);         % Use an INTEGRAL number of sine waves!!
Nlast=round(ncyc/(w*dt));
```

4. Allow time to pass for the system to settle.

```
u1=uicontrol(2,'Position',[10 10 150 30],'String','HIT to BREAK',...
    'Callback','stopval=1;''Enable','on','Value',5);
tic
while (toc<2) & (get(u1,'Value')>1) % settle time pause...
    drawnow
end
```

5. Set the mtrace trigger and capture the data stream.

```
mtrc31('SetFrame',[],1,0,N);
mtrc31('SetTrigger',y_addr(3,:),0,1);
mtrc31('LockProgram');
mtrc31('StartCapture');
while mtrc31('CaptureState')~=0
    drawnow;
end
my_data=mtrc31('FetchData');
% Take an INTEGRAL number of sine waves, total:
k=[0:(Nlast-1)]';
ncyc=floor(w*dt*Nlast);
y_out=my_data(2,1:Nlast);
y_in=my_data(1,1:Nlast);
t_out=dt*[1:Nlast];
mtrc31('UnlockProgram');
```

6. Apply the necessary equations to extract the estimates of gain and phase.

```
y_sin=((w*2*pi)*t_out);      % Ideal SINE at this freq
Bc=(2/(length(y_out)))*sum((y_out).*cos(y_sin));
Bs=(2/(length(y_out)))*sum((y_out).*sin(y_sin));
B_=sqrt(Bc^2+Bs^2);          % Amplitude of OUTPUT at this freq
Ac=(2/(length(y_out)))*sum((y_in).*cos(y_sin));
As=(2/(length(y_out)))*sum((y_in).*sin(y_sin));
A_=sqrt(Ac^2+As^2);          % Amplitude of INPUT at this freq (check)
Gain=B_/A_;
phi_out=atan2(Bc,Bs);
phi_in=atan2(Ac,As);
phi=phi_out-phi_in;      % Difference in phase (input->output) (rad)
```