Alternative Algorithm for Measuring Phase Differences in Two Shakers

Mike Garbus
Naval Surface Warfare Center Carderock Division
West Bethesda, Maryland
February 13, 2004

A system is currently in place to measure phase differences in two shakers. The existing system works well when the phase difference is between 0˚ and 100˚, and again between 230˚ and 360˚. Figure 1 shows the performance of the current system. In Figure 1, the red lines are the responses starting at the user defined phase offset going to the controllers response to that offset. If the controller did not correct the phase difference so that the shakers are within 15˚ of each other on the first attempt, the blue lines are the second attempt to correct the phase difference. Likewise, the yellow line is the third attempt if it was still not corrected.

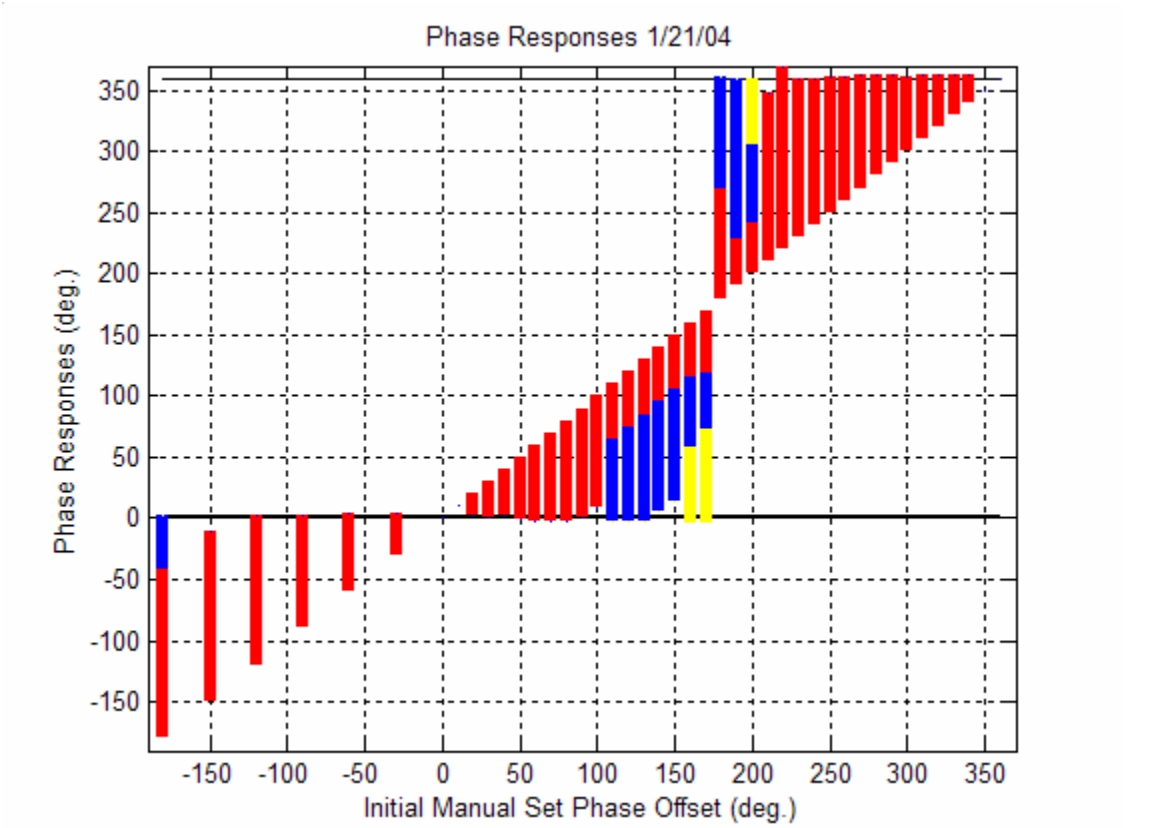

Figure 1. Results of current phasing system.

Although the performance of the current system was deemed acceptable, a new algorithm may be able to have much more accuracy in calculating the phase difference with more consistency. This paper is a brief informal derivation and discussion of an alternative method for calculating the phase difference in the shakers.

The derivation of this new algorithm makes four basic assumptions. First, the algorithm assumes that the noise on the two input signals is relatively small compared to the amplitude of the two signals. Secondly, the noise on the signals is white (i.e. its average energy is equally spread in the spectrum). Thirdly, the signals do not have a DC offset (i.e. they are AC coupled). And finally, the amplitude of the two signals is normalized.

Sinusoidal signals are often mathematically represented using phasors. A phasor is a complex number represented in a polar format. Let the two signals from the shakers be called f and g. Let's derive the phase difference first without noise on the signal for the sake of simplicity.

Let,

$$f = e^{i(\omega t+\phi_1)},$$
$$g = e^{i(\omega t+\phi_2)}, \text{ and}$$
$$h = \frac{f}{g} = \frac{e^{i(\omega t+\phi_1)}}{e^{i(\omega t+\phi_2)}} = e^{i(\phi_1-\phi_2)}.$$

Therefore, by Euler's formula, we see that,

$$h = e^{i(\phi_1-\phi_2)} = \cos(\phi_1-\phi_2)+i\cdot\sin(\phi_1-\phi_2).$$

Since we are only interested in the real component of *h*, we are only interested in,

$$h_{real} = \cos(\phi_1-\phi_2).$$

If the phase difference in *f* and *g* is constant (i.e. is not a function of time), then we would expect h_{real} to be constant. That is, h_{real} is a single constant number which is not dependant on time or any other value other than the difference in the phases of the two signals. This would imply that the phase difference is exactly,

$$\phi_1-\phi_2 = \cos^{-1}(h_{real}) = \cos^{-1}\left(\mathrm{Re}\left\{e^{i(\phi_1-\phi_2)}\right\}\right).$$

In the real signals, *f* and *g* have noise. The noise is white in nature and has the same properties on both signals. Since the noise is random, we can model it has having a time dependence. If the noise is additive white noise, *f* and *g* would become,

$$f(t) = e^{i(\omega t+\phi_1)} + N_1(t), \text{ and}$$
$$g(t) = e^{i(\omega t+\phi_2)} + N_2(t),$$

where $N_1(t)$ and $N_2(t)$ are the two additive white noise functions. The problem with modeling the signals this way is that when *f* and *g* are divided, you are not left with only a function of the phases. Instead, you are left with,

$$h = \frac{f(t)}{g(t)} = \frac{e^{i(\omega t+\phi_1)} + N_1(t)}{e^{i(\omega t+\phi_2)} + N_2(t)},$$

where the noise functions cannot be cancelled out of *h*. For this reason, it might be reasonable to think of the noise as being multiplied directly to the amplitude of each signal, such that,

$$e^{i(\omega t+\phi_1)} + N_1(t) = n_1(t) * e^{i(\omega t+\phi_1)}, \text{ and}$$
$$e^{i(\omega t+\phi_2)} + N_2(t) = n_2(t) * e^{i(\omega t+\phi_2)}.$$

In this case we would have,

$$f(t) = n_1(t) * e^{i(\omega t+\phi_1)},$$

$$g(t) = n_2(t) * e^{i(\omega t + \phi_2)}, \text{ and}$$

$$h = \frac{f(t)}{g(t)} = \frac{n_1(t) * e^{i(\omega t + \phi_1)}}{n_2(t) * e^{i(\omega t + \phi_2)}} = \left(\frac{n_1(t)}{n_2(t)}\right) \cdot e^{i(\phi_1 - \phi_2)} = \left(\frac{n_1(t)}{n_2(t)}\right) \cdot \left(\cos(\phi_1 - \phi_2) + i \cdot \sin(\phi_1 - \phi_2)\right).$$

Again, the real portion of *h* would become,

$$h_{real}(t) = \left(\frac{n_1(t)}{n_2(t)}\right) \cdot \cos(\phi_1 - \phi_2).$$

Since $n_1(t)$ and $n_2(t)$ are white in nature and have the same properties, averaging the quotient of the two at many points in time should equal unity, such that,

$$\left\langle \frac{n_1(t)}{n_2(t)} \right\rangle = 1.$$

For this reason, by averaging the quotient of the noise,

$$\langle h_{real}(t) \rangle = 1 \cdot \cos(\phi_1 - \phi_2).$$

Again, we find that with averaging,

$$\phi_1 - \phi_2 = \cos^{-1} \langle h_{real}(t) \rangle = \cos^{-1}\left(\mathrm{Re}\left\{e^{i(\phi_1 - \phi_2)}\right\}\right).$$

Therefore, we can obtain the same answer by considering multiple samples of the signals.

One way to actually implement this, which has been very successful in simulations, is to compute the quotient of $g(t)$ and $f(t)$ using linear regression. The two functions, $g(t)$ and $f(t)$, are nothing more than two *Mx1* matrices once they have been converted from an analog signal to a digital one. Using the least squares method of linear regression, the problem boils down to solving the following linear equation for *m*. In our case, *b* is set to zero.

$$f(a,b) = mg + b$$

Without showing the entire derivation of the least squares method of linear regression,

$$m = \frac{M\sum_{i=1}^{M} g_i f_i - \sum_{i=1}^{M} g_i \sum_{i=1}^{M} f_i}{M\sum_{i=1}^{M} g_i^2 - \left(\sum_{i=1}^{M} g_i\right)^2} = \frac{\left(\sum_{i=1}^{M} g_i f_i\right) - M \cdot \bar{g} \cdot \bar{f}}{\sum_{i=1}^{M} g_i^2 - M \cdot \bar{g}^2},$$

where *M* is the number of elements in *g* or *f*. Therefore, *h* is approximately equal to,

$$h_{approx} = m = \frac{\left(\sum_{i=1}^{M} g_i f_i\right) - M \cdot \bar{g} \cdot \bar{f}}{\sum_{i=1}^{M} g_i^2 - M \cdot \bar{g}^2}.$$

We can substitute h_{approx} into the derivation for the difference in phase to obtain,

$$\phi_1 - \phi_2 = \cos^{-1}\left(\mathrm{Re}\{h_{approx}\}\right).$$

MATLAB was used to evaluate the effectiveness of this method. MATLAB is ideal for simulating discrete, digital systems such as the shaker controller setup. In the MATLAB M-file, two signals were generated as defined by the user. Random white noise was added to the signals at a user specified average amplitude. The M-file was designed such that one of the signals is held at a constant phase angle and the other phase angle is swept, so you can evaluate the effectiveness of the algorithm at different phase angle differences. The noise amplitude is also swept so the algorithm can be evaluated at different noise levels. This information is compiled into a 3-dimensional graph. Figure 2 shows the algorithms ability to detect the phase difference in the two signals.

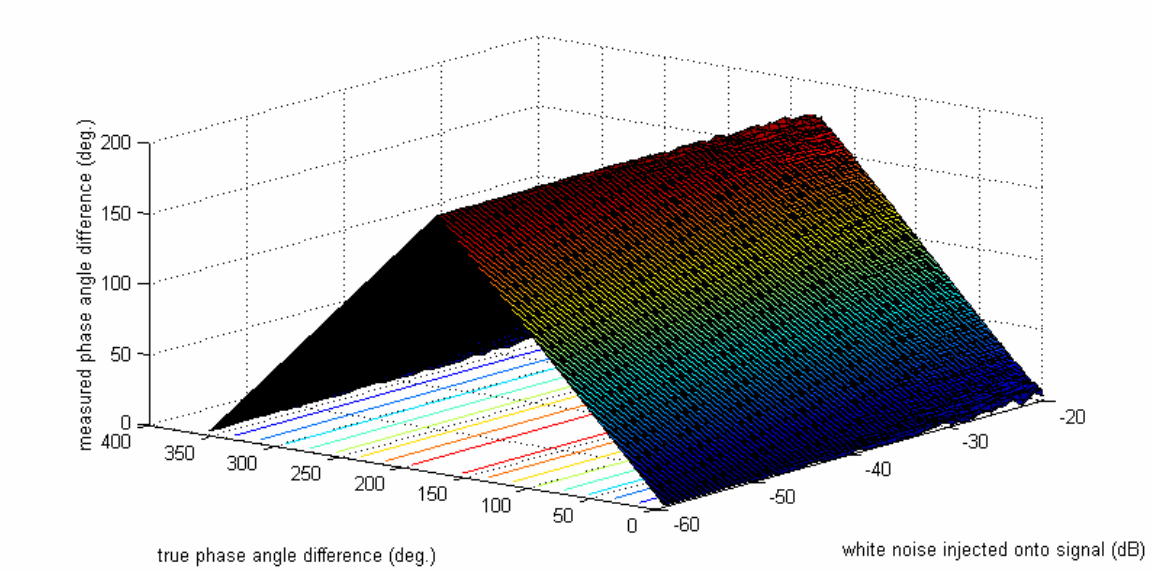

Figure 2. The algorithms phase difference response versus the actual phase difference and the noise amplitude.

As you can see from the graph, the algorithm works very well in the given noise range from 0˚ to 180˚. After 180˚, the algorithm begins to be unable to correctly measure the phase angle difference. The reason for this breakdown lays in the properties of the arccosine. The arccosine is an even function, which implies that the magnitude of the result will be symmetric about 180˚. For example, the algorithm has trouble differentiating between phase differences such as 160˚ and 200˚, because they are both 20˚ away from 180˚. Figure 3, which is the error in the in the algorithm, also reflects this inability. Note that all of the points that are shown in Figure 3 are within 15% of being correct. Over 0˚ to 180˚, there is virtually no error other than near 0˚, especially at lower noise levels. This figure shows that the algorithm works extremely well from 0˚ to 180˚.

It should also be noted that even though the measured difference may be within 5˚ of the actual phase difference when the actual phase difference is near 0˚, the error will appear larger than at greater phase differences because the error is evaluated using a percentage. Figure 4 shows the difference between the actual phase difference and the measured phase difference. This graph shows that errors near 0˚ may not be a severe as they appear in Figure 3.

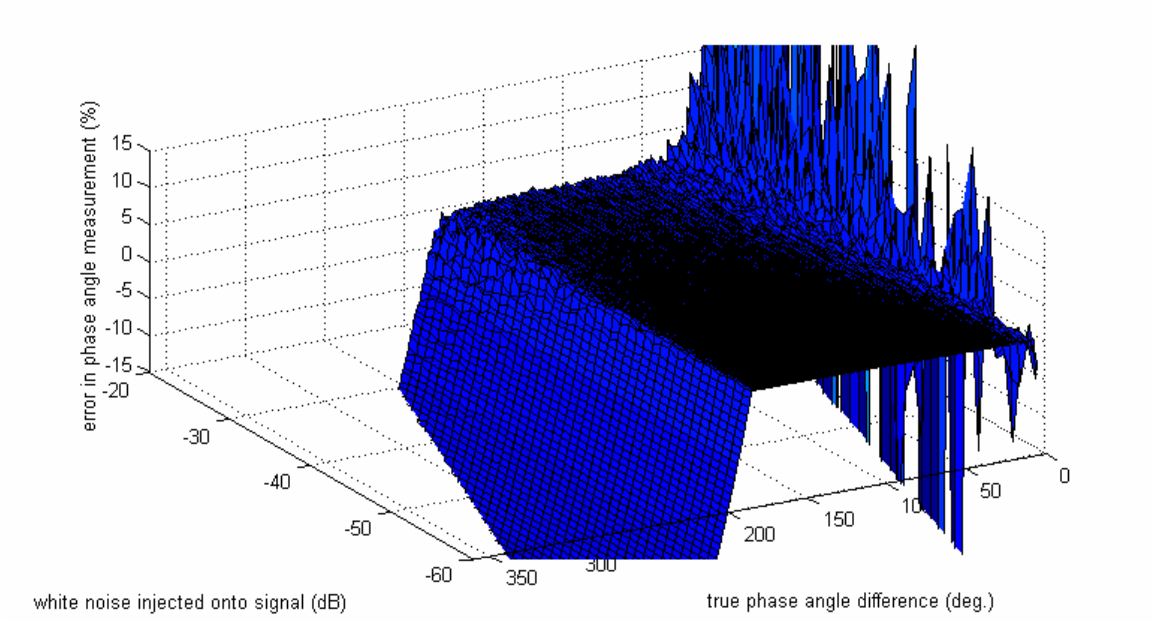

Figure 3. Error between measured phase difference and the actual phase difference in the signals.

The following equations were used to create Figures 3 and 4, respectively, at each point.

$$error = 100 \cdot \frac{(actual\ \ phase) - (measured\ \ phase)}{(actual\ \ phase)}$$

$$difference = (actual\ \ phase) - (measured\ \ phase)$$

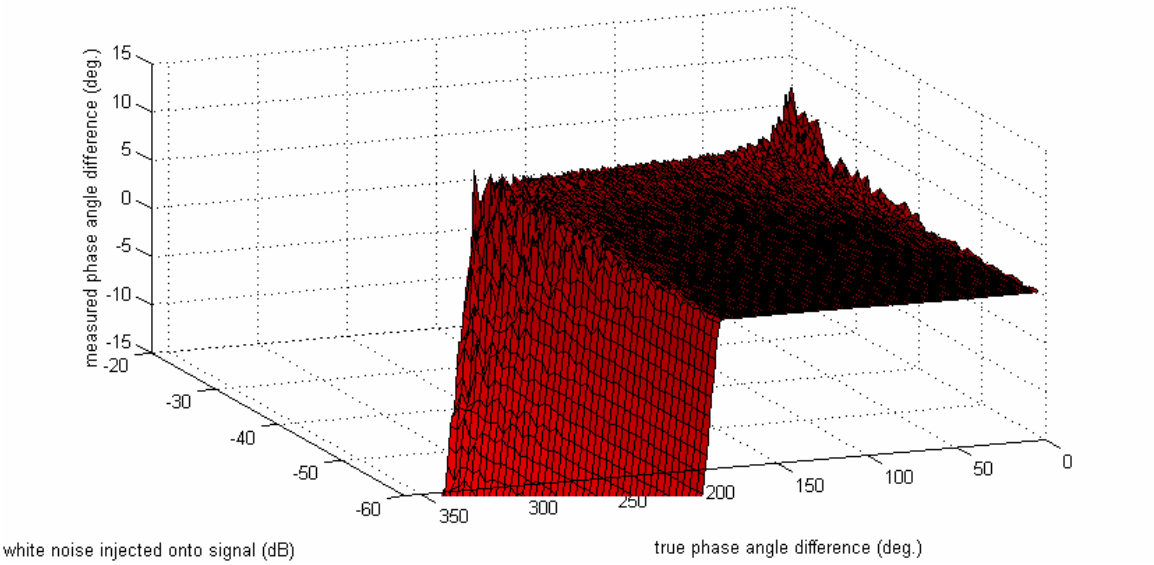

Figure 4. Difference between measured phase and actual phase.

You can see that between 0˚ and 180˚ in Figure 4, the measured phase difference is approximately only 7˚ off in the worst conditions. At normal operating conditions, the measured phase difference is less than 2˚ off. Figures 2, 3, and 4 were generated by the following MATLAB M-file.

MATLAB Simulation of Alternative Phase Difference Measuring Method 1

```
% Alternate Phase Difference Measurement Error
%*******************************************
% This MATLAB code simulates an algorithm for
% measuring the phase difference between two
% noisy sinusodial signals with the same
% frequency after they are read by the ADC on
% the PMAC.
%*******************************************
% Keeps one sinusoidal wave constant and
% varies the other's phase.  It also varies
% the noise on each signal with the same dB
% level on both signals.  You can
% also take multiple readings for all points
% that are averaged at each setting controlled
% by the number of averages.  Basically, it
% sweeps through phase angles for the second
% sinusoidal wave and sweeps through noise
% levels on both waves.
%*******************************************



% USER DEFINED VARIABLES
%-----------------------
frequency = 20; %Hertz
sampling_rate = 1000; %samples/sec
number_of_seconds = 1; %sample length in seconds
phase1_deg = 0; %in degrees (constant)
%NOTE: start phase & noise must be LESS THAN end phase & noise
startphase2_deg = 0; %in degrees
endphase2_deg = 360; %in degrees
startnoisedB = -60; %in dB
endnoisedB = -20; %in dB
number_of_averages = 1; %must be greater than or equal to 1

% define matrices and arrays
%---------------------------
zdata_measuredphase=zeros(abs(endphase2_deg-startphase2_deg),abs(endnoisedB-startnoisedB));
zdata_error=zdata_measuredphase;
zdata_difference=zdata_measuredphase;
ydata_noise=zeros(length(zdata_error(1,:)),1);
xdata_phase=zeros(length(zdata_error(:,1)),1);
t=[0:(1/sampling_rate):number_of_seconds];

%one time calculations
%---------------------
phase1_rad = phase1_deg*pi/180;
omega=2*pi*frequency;

%loops
%-----
for phase2_deg = 1:length(xdata_phase) %1 deg increments
   xdata_phase(phase2_deg)=phase2_deg+startphase2_deg-phase1_deg;
   phase2_rad = (phase2_deg+startphase2_deg)*pi/180;
   for noise_dB = 1:length(ydata_noise) %1 dB increments
      ydata_noise(noise_dB)=noise_dB+startnoisedB;
      noise_linear = 10^((noise_dB+startnoisedB)/20);
      average_summer=0;
      for i=1:number_of_averages
         f=cos(omega*t+phase1_rad)+(noise_linear*2*rand(size(t)));
         g=cos(omega*t+phase2_rad)+(noise_linear*2*rand(size(t)));

         mean_f=mean(f);
         mean_g=mean(g);
         sum_f=sum(f);
```

```
        sum_g=sum(g);
        sum_gf=sum(g.*f);
        sum_g2=sum(g.*g);
        m=(sum_gf-sum_g*sum_f/length(f))/(sum_g2-sum_g^2/length(f));
        h=m;


        measured_phase_diff=acos(h)*180/pi;
        average_summer=average_summer+measured_phase_diff;
      end
      zdata_measuredphase(phase2_deg,noise_dB)=real(average_summer/number_of_averages);
      if (xdata_phase(phase2_deg)~=0)
        zdata_error(phase2_deg,noise_dB)=(zdata_measuredphase(phase2_deg,noise_dB)-
xdata_phase(phase2_deg))/xdata_phase(phase2_deg)*100;
      else
        zdata_error(phase2_deg,noise_dB)=NaN;
      end
      zdata_difference(phase2_deg,noise_dB)=zdata_measuredphase(phase2_deg,noise_dB)-xdata_phase(phase2_deg);
   end
end


%Plot Data
%-----------
figure(2)
surfc(ydata_noise,xdata_phase,zdata_measuredphase)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('measured phase angle difference (deg.)')

figure(3)
surfc(ydata_noise,xdata_phase,zdata_error)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('error in phase angle measurement (%)')
axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);

figure(4)
surfc(ydata_noise,xdata_phase,zdata_difference)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('measured phase angle difference (deg.)')
axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);
```

From Figure 2, we see that it is expected that the algorithm follows,

$$\theta_{out} = -|\theta_{in} - 180| + 180 \quad for\ 0^\circ \le \theta_{in} \le 360^\circ,$$

where θ_{in} is the actual difference in the two signals and θ_{out} is the measured difference in the two signals according to the algorithm. Obviously after 180˚, the measured difference is incorrect. It should be noted that the pattern repeats at increments of 360˚. In order to circumvent the algorithms inability to accurately detect the phase difference after 180˚, a second run can be taken where the algorithm induces an additional phase offset in one of the signals such that,

$$f_2 = e^{i(\omega t+\phi_1)},$$
$$g_2 = e^{i(\omega t+\phi_2+\Delta)},$$

where f_2 and g_2 are the signals from the second run, and Δ is the induced phase offset. Comparing the data from the first and second run, accurate readings can be calculated between 180˚ and 360˚, if $\Delta < 90^\circ$. Let us define the following variables:

1. θ_1 is the difference angle measured in the first run.
2. θ_2 is the difference angle measured in the second run.
3. Δ is the angular offset induced in the second run (as defined above).
4. δ is the maximum allowable error in the measurement (in degrees).
5. θ_{final} is the final angular difference based on all of the above variables.

The following algorithm can be used to compare the data from the first run and the second run, so that accurate measurements can be made over the entire range from 0˚ to 360˚.

IF $(\Delta \le \theta_1 \le 180 - \Delta)$
    IF $(\theta_1 + \Delta - \delta < \theta_2 < \theta_1 + \Delta + \delta)$
        $\theta_{final} = \theta_1$
    ELSEIF $(\theta_1 - \Delta - \delta < \theta_2 < \theta_1 - \Delta + \delta)$
        $\theta_{final} = 360 - \theta_1$
    ELSE
        Display ERROR CODE 1
ELSEIF $(\theta_1 > 180 - \Delta)$
    IF $(360 - \theta_1 - \Delta - \delta < \theta_2 < 360 - \theta_1 - \Delta + \delta)$
        $\theta_{final} = \theta_1$
    ELSEIF $(\theta_1 - \Delta - \delta < \theta_2 < \theta_1 - \Delta + \delta)$
        $\theta_{final} = 360 - \theta_1$
    ELSE
        Display ERROR CODE 2
ELSEIF $(\theta_1 < \Delta)$
    IF $(\theta_1 + \Delta - \delta < \theta_2 < \theta_1 + \Delta + \delta)$
        $\theta_{final} = \theta_1$
    ELSEIF $(-\theta_1 + \Delta - \delta < \theta_2 < -\theta_1 + \Delta + \delta)$
        $\theta_{final} = 360 - \theta_1$
    ELSE
        Display ERROR CODE 3
ELSE
    Display ERROR CODE 4

Taking two runs with the second run having an induced offset, the above algorithm was implemented in MATLAB. Similar to Figures 2, 3, and 4, Figures 5, 6, and 7 show the outcome of the algorithm. Figures 5, 6, and 7 were produced with $\Delta = 45^\circ$ and $\delta = 10^\circ$.

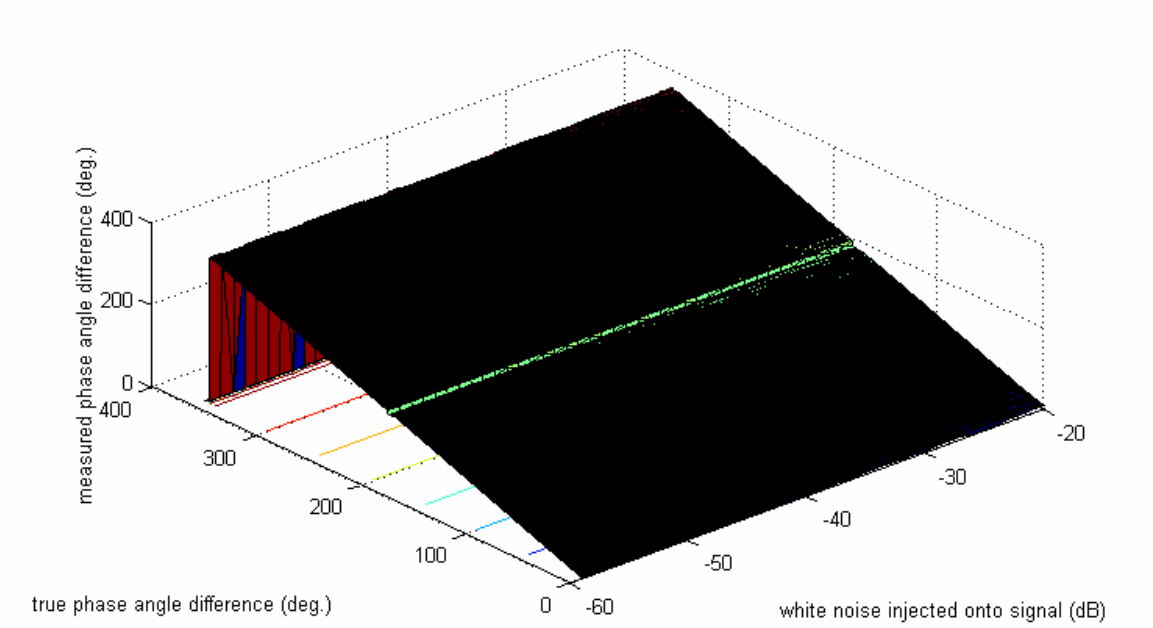

Figure 5. The algorithms phase difference response versus the actual phase difference and the noise amplitude after second run was taken with induced offset.

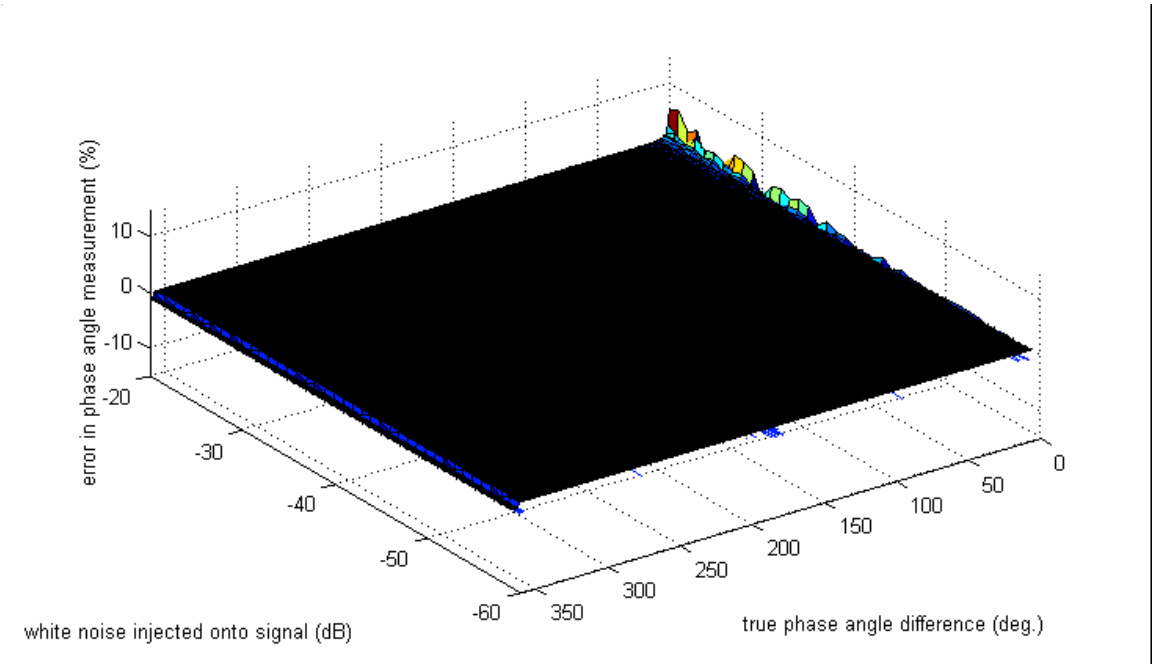

Figure 6. Error between measured phase difference and the actual phase difference in the signals after second run was taken with induced offset.

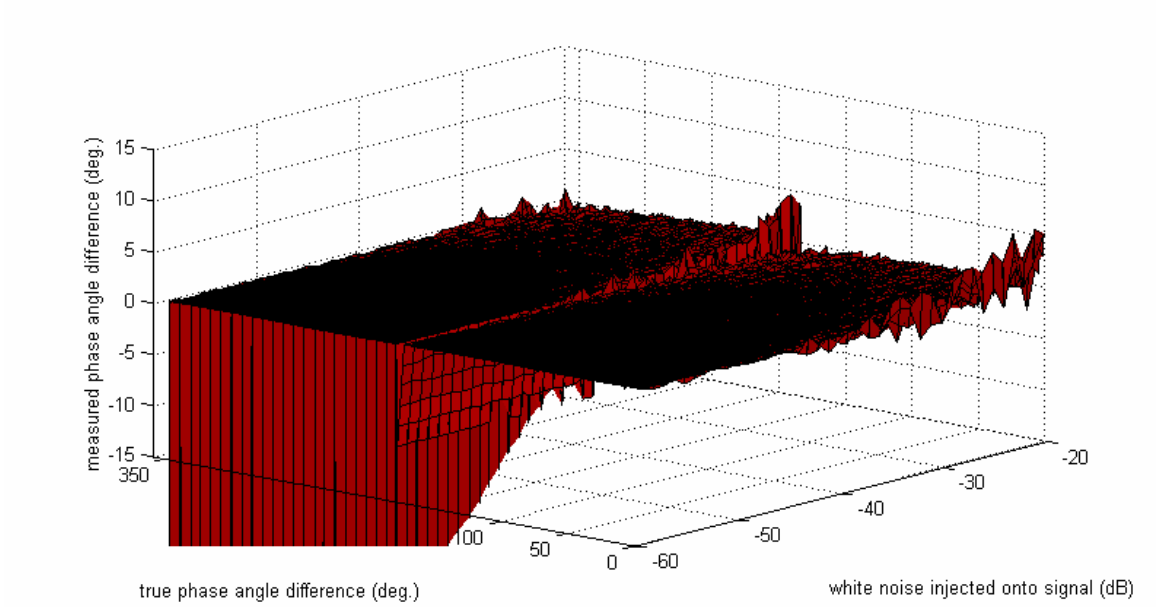

Figure 7. Difference between measured phase and actual phase after second run was taken with induced offset.

As you can see from Figure 5, the algorithm can now accurately detect phase differences above 180˚. Figure 6 shows that the error is normally less than 1%, and only rises above that near 0˚ at higher noise levels. Figure 7 shows that the algorithm is always within 10˚ of the correct answer. The largest deviation from the correct answer occurs near 180˚, where the answer is about 10˚ off. Near 0˚, there is some deviation, but it is within 5˚ of the correct answer. Otherwise, the answer is very accurate.

Comparing this data with the data from the phase difference measuring system that is currently being used (data in Figure 1), we can easily see that the new alternative algorithm performs much better between 100˚ and 230˚. The new alternative algorithm is always within 15˚ of the actual phase difference; where as the current system is not always within 15˚. From this preliminary simulation, it appears the new alternative algorithm outperforms the current system.

The MATLAB code used to produce Figures 5, 6, and 7 is shown below.

MATLAB Simulation of Alternative Phase Difference Measuring Method Complete

```
% Alternate Phase Difference Measurement Error
%*****************************************
% This MATLAB code simulates an algorithm for
% measuring the phase difference between two
% noisy sinusodial signals with the same
% frequency after they are read by the ADC on
% the PMAC.
%*****************************************
% RUN 1
% Keeps one sinusoidal wave constant and
% varies the other's phase.  It also varies
% the noise on each signal with the same dB
% level on both signals.  You can
% also take multiple readings for all points
% that are averaged at each setting controlled
% by the number of averages.  Basically, it
% sweeps through phase angles for the second
% sinusoidal wave and sweeps through noise
% levels on both waves.
```

```
%*******************************************
% RUN 2
% Repeats run one with an induced offset
%*******************************************
% Performs analysis on data from RUN 1 and 2
%*******************************************
%*******************************************

close all
clear all


% USER DEFINED VARIABLES
%----------------------
frequency = 20; %Hertz
sampling_rate = 1000; %samples/sec
number_of_seconds = 1; %sample length in seconds
phase1_deg = 0; %in degrees (constant)
%NOTE: start phase & noise must be LESS THAN end phase & noise
startphase2_deg = 0; %in degrees
endphase2_deg = 360; %in degrees
startnoisedB = -60; %in dB
endnoisedB = -20; %in dB
number_of_averages = 1; %must be greater than or equal to 1
angle_offset=45;%degrees added to second run
allowable_error=10;%degrees allowable off

% define matrices and arrays
%---------------------------
zdata_measuredphase=zeros(abs(endphase2_deg-startphase2_deg),abs(endnoisedB-startnoisedB));
zdata_error=zdata_measuredphase;
zdata_difference=zdata_measuredphase;
ydata_noise=zeros(length(zdata_error(1,:)),1);
xdata_phase=zeros(length(zdata_error(:,1)),1);
t=[0:(1/sampling_rate):number_of_seconds];

%one time calculations
%---------------------
phase1_rad = phase1_deg*pi/180;
omega=2*pi*frequency;

%loops
%-----
for phase2_deg = 1:length(xdata_phase) %1 deg increments
   xdata_phase(phase2_deg)=phase2_deg+startphase2_deg-phase1_deg;
   phase2_rad = (phase2_deg+startphase2_deg)*pi/180;
   for noise_dB = 1:length(ydata_noise) %1 dB increments
      ydata_noise(noise_dB)=noise_dB+startnoisedB;
      noise_linear = 10^((noise_dB+startnoisedB)/20);
      average_summer=0;
      for i=1:number_of_averages
         f=cos(omega*t+phase1_rad)+(noise_linear*2*rand(size(t)));
         g=cos(omega*t+phase2_rad)+(noise_linear*2*rand(size(t)));

         mean_f=mean(f);
         mean_g=mean(g);
         sum_f=sum(f);
         sum_g=sum(g);
         sum_gf=sum(g.*f);
         sum_g2=sum(g.*g);
         m=(sum_gf-sum_g*sum_f/length(f))/(sum_g2-sum_g^2/length(f));
         h=m;


         measured_phase_diff=acos(h)*180/pi;
         average_summer=average_summer+measured_phase_diff;
      end
      zdata_measuredphase(phase2_deg,noise_dB)=real(average_summer/number_of_averages);
      if (xdata_phase(phase2_deg)~=0)
         zdata_error(phase2_deg,noise_dB)=(zdata_measuredphase(phase2_deg,noise_dB)-
xdata_phase(phase2_deg))/xdata_phase(phase2_deg)*100;
      else
         zdata_error(phase2_deg,noise_dB)=NaN;
      end
```

```
      zdata_difference(phase2_deg,noise_dB)=zdata_measuredphase(phase2_deg,noise_dB)-xdata_phase(phase2_deg);
   end
end


%Plot Data for Single Run
%------------------------
% figure(2)
% surfc(ydata_noise,xdata_phase,zdata_measuredphase)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('measured phase angle difference (deg.)')
%
% figure(3)
% surfc(ydata_noise,xdata_phase,zdata_error)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('error in phase angle measurement (%)')
% axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);
%
% figure(4)
% surfc(ydata_noise,xdata_phase,zdata_difference)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('measured phase angle difference (deg.)')
% axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);


%--------------------------
% Second Run
%--------------------------

% define matrices and arrays
%---------------------------
zdata_measuredphase_2=zeros(abs(endphase2_deg-startphase2_deg),abs(endnoisedB-startnoisedB));
zdata_error_2=zdata_measuredphase;
zdata_difference_2=zdata_measuredphase;
ydata_noise_2=zeros(length(zdata_error(1,:)),1);
xdata_phase_2=zeros(length(zdata_error(:,1)),1);

%loops
%-----
for phase2_deg = 1:length(xdata_phase) %1 deg increments
   xdata_phase_2(phase2_deg)=phase2_deg+startphase2_deg-phase1_deg;
   phase2_rad = (phase2_deg+startphase2_deg)*pi/180;
   for noise_dB = 1:length(ydata_noise_2) %1 dB increments
      ydata_noise_2(noise_dB)=noise_dB+startnoisedB;
      noise_linear = 10^((noise_dB+startnoisedB)/20);
      average_summer=0;
      for i=1:number_of_averages
         f=cos(omega*t+phase1_rad)+(noise_linear*2*rand(size(t)));
         g=cos(omega*t+phase2_rad+angle_offset/180*pi)+(noise_linear*2*rand(size(t)));

         mean_f=mean(f);
         mean_g=mean(g);
         sum_f=sum(f);
         sum_g=sum(g);
         sum_gf=sum(g.*f);
         sum_g2=sum(g.*g);
         m=(sum_gf-sum_g*sum_f/length(f))/(sum_g2-sum_g^2/length(f));
         h=m;


         measured_phase_diff=acos(h)*180/pi;
         average_summer=average_summer+measured_phase_diff;
      end
      zdata_measuredphase_2(phase2_deg,noise_dB)=real(average_summer/number_of_averages);
      if (xdata_phase_2(phase2_deg)~=0)
         zdata_error_2(phase2_deg,noise_dB)=(zdata_measuredphase(phase2_deg,noise_dB)-
xdata_phase(phase2_deg))/xdata_phase(phase2_deg)*100;
      else
         zdata_error_2(phase2_deg,noise_dB)=NaN;
      end
      zdata_difference_2(phase2_deg,noise_dB)=zdata_measuredphase_2(phase2_deg,noise_dB)-xdata_phase_2(phase2_deg);
```

```
    end
end

%Plot Data for Second Single Run
%--------------------------------
% figure(5)
% surfc(ydata_noise_2,xdata_phase_2,zdata_measuredphase_2)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('measured phase angle difference (deg.)')
%
% figure(6)
% surfc(ydata_noise_2,xdata_phase_2,zdata_error_2)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('error in phase angle measurement (%)')
% axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);
%
% figure(7)
% surfc(ydata_noise_2,xdata_phase_2,zdata_difference_2)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('measured phase angle difference (deg.)')
% axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);
%
%
% figure(8)
% surfc(ydata_noise,xdata_phase,zdata_measuredphase)
% hold on
% surfc(ydata_noise_2,xdata_phase_2,zdata_measuredphase_2)
% ylabel('true phase angle difference (deg.)')
% xlabel('white noise injected onto signal (dB)')
% zlabel('measured phase angle difference (deg.)')
% hold off


%------------------------------
%Analysis of two measurements
%------------------------------
zfinal=zeros(abs(endphase2_deg-startphase2_deg),abs(endnoisedB-startnoisedB));


for i=1:length(zdata_measuredphase(:,1))
   for j=1:length(zdata_measuredphase(1,:))
      if ((zdata_measuredphase(i,j) <= (180 - angle_offset))&(zdata_measuredphase(i,j) >= (angle_offset)))
         if ((zdata_measuredphase_2(i,j)>(zdata_measuredphase(i,j)+angle_offset-allowable_error)) & ...
               (zdata_measuredphase_2(i,j)<(zdata_measuredphase(i,j)+angle_offset+allowable_error)))
            zfinal(i,j)=zdata_measuredphase(i,j);
            %fprintf('a');
         elseif ((zdata_measuredphase_2(i,j)>(zdata_measuredphase(i,j)-angle_offset-allowable_error)) & ...
               (zdata_measuredphase_2(i,j)<(zdata_measuredphase(i,j)-angle_offset+allowable_error)))
            zfinal(i,j)=360-zdata_measuredphase(i,j);
            %fprintf('b');
         else
            fprintf('\n180-30<theta1 %i %i',i,j);
         end
      elseif (zdata_measuredphase(i,j) > (180 - angle_offset))
         if ((zdata_measuredphase_2(i,j)<(360-angle_offset+allowable_error-zdata_measuredphase(i,j))) & ...
               (zdata_measuredphase_2(i,j)>(360-angle_offset-allowable_error-zdata_measuredphase(i,j))))
            zfinal(i,j)=zdata_measuredphase(i,j);
            %fprintf('c');
         elseif ((zdata_measuredphase_2(i,j)>(zdata_measuredphase(i,j)-angle_offset-allowable_error)) & ...
               (zdata_measuredphase_2(i,j)<(zdata_measuredphase(i,j)-angle_offset+allowable_error)))
            zfinal(i,j)=360-zdata_measuredphase(i,j);
            %fprintf('d');
         else
            fprintf('\n180-30>theta1 %i %i',i,j);
         end
      elseif (zdata_measuredphase(i,j) < (angle_offset))
         if ((zdata_measuredphase_2(i,j)>(zdata_measuredphase(i,j)+angle_offset-allowable_error)) & ...
               (zdata_measuredphase_2(i,j)<(zdata_measuredphase(i,j)+angle_offset+allowable_error)))
            zfinal(i,j)=zdata_measuredphase(i,j);
            %fprintf('e');
         elseif ((zdata_measuredphase_2(i,j)>(angle_offset-zdata_measuredphase(i,j)-allowable_error)) & ...
```

```
                (zdata_measuredphase_2(i,j)<(angle_offset-zdata_measuredphase(i,j)+allowable_error)))
            zfinal(i,j)=360-zdata_measuredphase(i,j);
            %fprintf('f');
          end
       else
          fprintf('Error 1')
       end
    end
    %fprintf('\n')
end


%---------------------------------
%Plot Final Data
%---------------------------------
zfinal_actual=zeros(size(zfinal));
for k=1:length(zfinal(1,:))
    zfinal_actual(:,k)=xdata_phase(:);
end

figure(9)
surfc(ydata_noise,xdata_phase,zfinal)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('measured phase angle difference (deg.)')

figure(10)
surfc(ydata_noise_2,xdata_phase_2,(zfinal-zfinal_actual)./zfinal_actual)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('error in phase angle measurement (%)')
axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);

figure(11)
surfc(ydata_noise_2,xdata_phase_2,zfinal-zfinal_actual)
ylabel('true phase angle difference (deg.)')
xlabel('white noise injected onto signal (dB)')
zlabel('measured phase angle difference (deg.)')
axis([startnoisedB,endnoisedB,startphase2_deg,endphase2_deg,-15,15]);
```

The following is the basic pseudo-code for the complete alternative algorithm.

define $\delta = 10^\circ$ *(user should have ability to override default)*

define $\Delta = 45^\circ$ *(user should have ability to override default)*

Take Initial Data from $f_1(t)$ *and* $g_1(t)$

Subtract DC offset

Normalize Data

$M = \#$ *of data points in* $f_1(t)$ *and* $g_1(t)$

$$sum\_f = \sum_{i=1}^{M} f_{1,i}$$

$$sum\_g = \sum_{i=1}^{M} g_{1,i}$$

$$sum\_gf = \sum_{i=1}^{M} g_{1,i} f_{1,i}$$

$$sum\_gg = \sum_{i=1}^{M} g_{1,i}^2$$

$$h = \frac{M \cdot (sum\_gf) - (sum\_g) \cdot (sum\_f)}{M \cdot (sum\_gg) - (sum\_g)^2}$$

$$h\_real = \mathrm{Re}\{h\}$$

$$\theta_1 = \cos^{-1}(h\_real)$$

Add offset, Δ, *to* g

Take Second Data from $f_2(t)$ *and* $g_2(t)$

Subtract DC offset

Normalize Data

$M = \#$ *of data points in* $f_2(t)$ *and* $g_2(t)$

$$sum\_f = \sum_{i=1}^{M} f_{2,i}$$

$$sum\_g = \sum_{i=1}^{M} g_{2,i}$$

$$sum\_gf = \sum_{i=1}^{M} g_{2,i} f_{2,i}$$

$$sum\_gg = \sum_{i=1}^{M} g_{2,i}^2$$

$$h = \frac{M \cdot (sum\_gf) - (sum\_g) \cdot (sum\_f)}{M \cdot (sum\_gg) - (sum\_g)^2}$$

$$h\_real = \mathrm{Re}\{h\}$$

$$\theta_2 = \cos^{-1}(h\_real)$$

continue on next page….

IF $(\Delta \le \theta_1 \le 180 - \Delta)$

    IF $(\theta_1 + \Delta - \delta < \theta_2 < \theta_1 + \Delta + \delta)$

        $\theta_{final} = \theta_1$

    ELSEIF $(\theta_1 - \Delta - \delta < \theta_2 < \theta_1 - \Delta + \delta)$

        $\theta_{final} = 360 - \theta_1$

    ELSE

        Display ERROR CODE 1

ELSEIF $(\theta_1 > 180 - \Delta)$

    IF $(360 - \theta_1 - \Delta - \delta < \theta_2 < 360 - \theta_1 - \Delta + \delta)$

        $\theta_{final} = \theta_1$

    ELSEIF $(\theta_1 - \Delta - \delta < \theta_2 < \theta_1 - \Delta + \delta)$

        $\theta_{final} = 360 - \theta_1$

    ELSE

        Display ERROR CODE 2

ELSEIF $(\theta_1 < \Delta)$

    IF $(\theta_1 + \Delta - \delta < \theta_2 < \theta_1 + \Delta + \delta)$

        $\theta_{final} = \theta_1$

    ELSEIF $(-\theta_1 + \Delta - \delta < \theta_2 < -\theta_1 + \Delta + \delta)$

        $\theta_{final} = 360 - \theta_1$

    ELSE

        Display ERROR CODE 3

ELSE

    Display ERROR CODE 4

Subtract θ_{final} *from the phase of* g