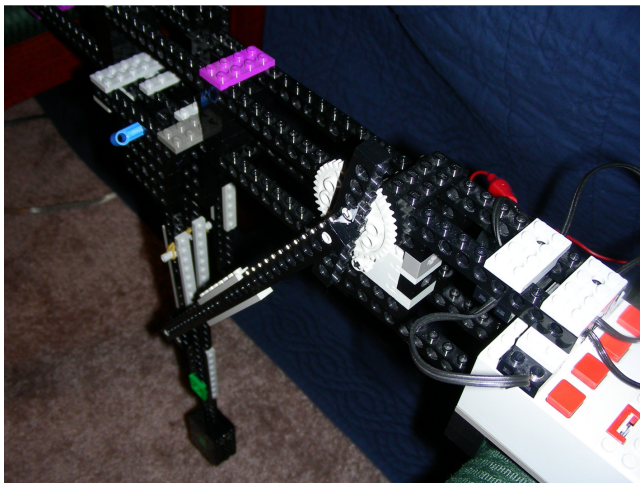
Analysis of a Nonlinear Duffing-Like LEGO Oscillator



Kevin Claytor

Physics 213

Final Project

12/10/2010

1. Introduction

The Duffing oscillator is one of the standard tools of any nonlinear dynamics toolbox, as it serves as an example system which can be applied to many situations. The model demonstrates such effects as damping, stiffening springs, and hysteresis. This project is interested in developing a model very similar to the Duffing system using the modular LEGO building blocks – hence this system can be used to demonstrate the Duffing oscillator in lecture one week, and then it can be re-built to demonstrate another nonlinear effect in a following week.

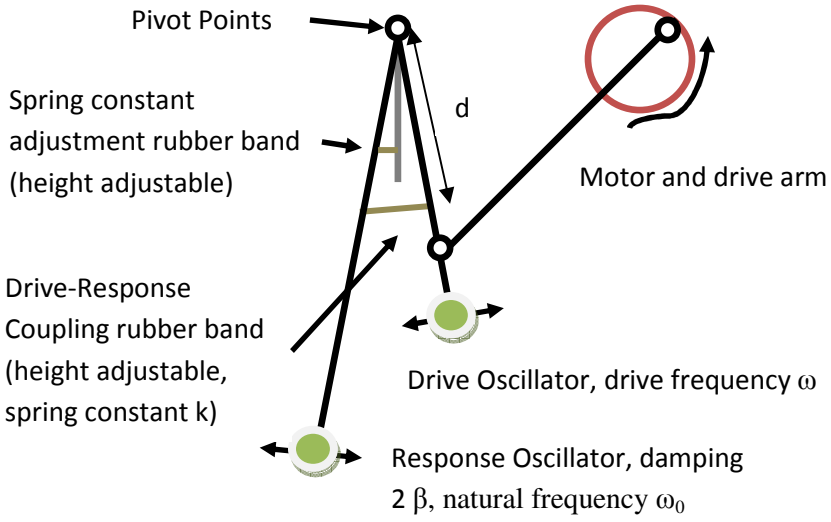


Figure 1: A schematic of the oscillator, showing the primary (response) oscillator (or pendulum), the drive oscillator, how they are connected, and how the response pendulum's restoring force can be made nonlinear and adjusted.

Figure 1 shows a schematic of the oscillator, which was designed so that nearly all of the phase space could be mapped out. The drive arm is powered by a variable speed motor, allowing for the frequency of oscillation to be varied. By adjusting where the coupling bar is on the motor, the amplitude of the drive could be adjusted. The coupling strength of the drive arm to the response (center) arm could be varied by adjusting the height of the pin holding the rubber band connecting the two was located. Finally, a rubber band could attach the response pendulum to the fixed arm, creating additional restoring force for larger displacements. That is, the spring constant can be adjusted from a nearly linear model, to a stiffening spring model. The rest of this paper will be concerned with exploring this phase space. However, due to time constraints only the coupling constant between the drive and response arms will be examined in detail.

As shown in Figure 1, the driving force is not just a torque about the pivot point of the oscillator, instead it is provided by another drive pendulum swinging back and forth, which is then connected by a rubber band. So instead of the usual Duffing equation, the new equation of motion reflects the fact that the force on the pendulum depends on the length of the rubber

band (which maps out a cord of a circle) connecting the two oscillators. Our equation of motion now becomes;

$$\ddot{\theta_r} - 2\beta\dot{\theta_r} - \omega^2 Sin(\theta_r) = 2\ D\ Sin(\frac{\theta_d - \theta_r}{2}) \quad (1)$$

Please refer to equation 1 for the variable definitions. We add here that θ_d, and θ_r are the angles of the drive and response arms, and D = d k / m l^2, ie; the effective coupling strength d*k (since both are varied to vary the connection force) divided by the moment of inertia of the oscillator. θ_d as a function of time is known to be just A*Sin(ω t), where A is the amplitude of the drive oscillator. The sine on the right hand side can be expanded via the sum of angle identities, and for small angles of θ_r and for small A we obtain;

$$\ddot{\theta_r} - 2\beta\dot{\theta_r} - (\omega^2 - D)\theta_r = D\ A\ Sin(\omega\ t) \quad (2)$$

These two equations will be the focus of our experimental study.

2. Experimental Method

Figure 2 shows the experimental setup. The LEGO oscillator was suspended and allowed to hang freely. A drive oscillator was connected to a standard LEGO motor (geared to provide maximum torque), which was powered by two 9 V LEGO battery packs wired in parallel. A 10 kΩ resistor wired in series provided a variable speed control, hence the voltage drop across the motor and resistor would sum to 18 V. A voltmeter was used to measure the voltage drop across the resistor as a gauge of how much voltage was provided to the motor, allowing for consistent motor speeds and intervals to be adjusted.

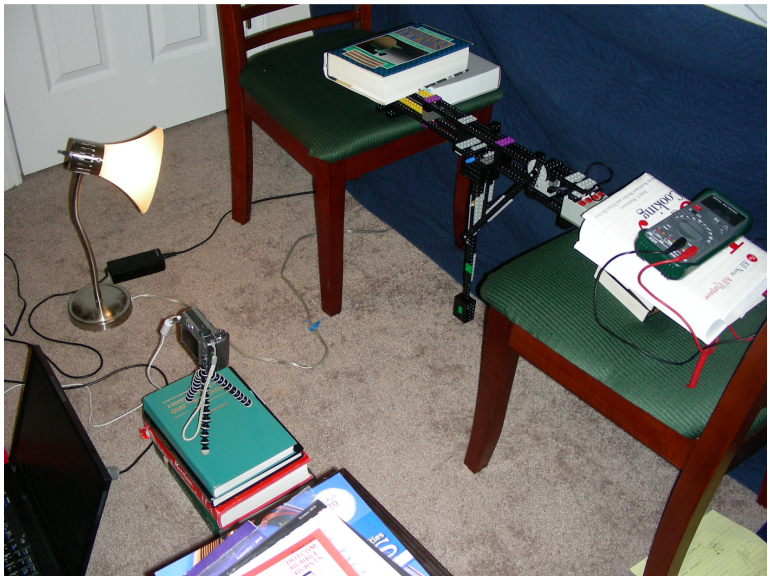
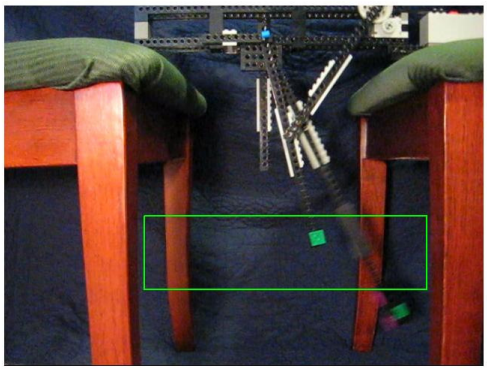


Figure 2: The experimental setup. A digital camera (center) recorded the position of the oscillator and drive mechanism. The voltmeter measures the potential drop across a variable resistor used to control the speed of the drive arm. The drive arm hangs closest to the camera, while the pendulum arm is the longest, and hangs in the middle. Behind the primary oscillator arm, the fixed arm resides. The books provide stabilization of the mechanism at higher frequencies.

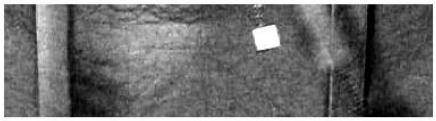
A digital camera was placed perpendicular to the oscillation plane and roughly at the midpoint of the oscillator. A blue cloth was draped behind the pendulum to provide additional contrast, and a light illuminate the pendulum from the front, allowing for easier tracking.

Data was taken by increasing the voltage to the motor until stable oscillation was reached. A 10 second video was recorded and the speed of the motor was increased again. This process was repeated through the resonance of the oscillator, and then the motor speed was backed down, still recording 10 s clips of video when the oscillation magnitude had reached the steady state.



A)

B)



C)

D)

Figure 3: The image analysis technique. The first frame (A) shows the full image frame, and the region that was selected as the drive oscillator region of interest. (B) is only the green channel of the region of interest, which then has its high and

low limits stretched (C), and finally a 99% threshold selects only the greenest pixels in the channel (D), the tracking dot.

Figure 3 illustrates the data analysis algorithm, which can be found in Appendix A. Fixed to the end of both the drive arm and the response pendulum arm was a green LEGO that provided enough contrast from the blue and red background that it could be easily located The process to do this location is as follows. The video was loaded into MATLAB and the user input three parameters. First a point was selected that defined the pivot point of the oscillator. Second, a region was defined where the driven pendulum would be. Finally, a region was selected where the drive pendulum was located. The algorithm then went through each frame, pulling out the region of drive and response, selecting the green channel, and applying a 99% threshold. The green dot now appears as a block of 1's on a matrix of 0's – the average of the locations then gave the location of the dot, and hence the pendulum arm. This was converted into an angle by taking the arc-tangent of the difference in coordinates of the arm from the pivot point. The results of this method are illustrated in Figure 4. Once a time series of the pendulum was recorded, Mathematica was used to fit the data to a sine wave (code for this is found in Appendix B).

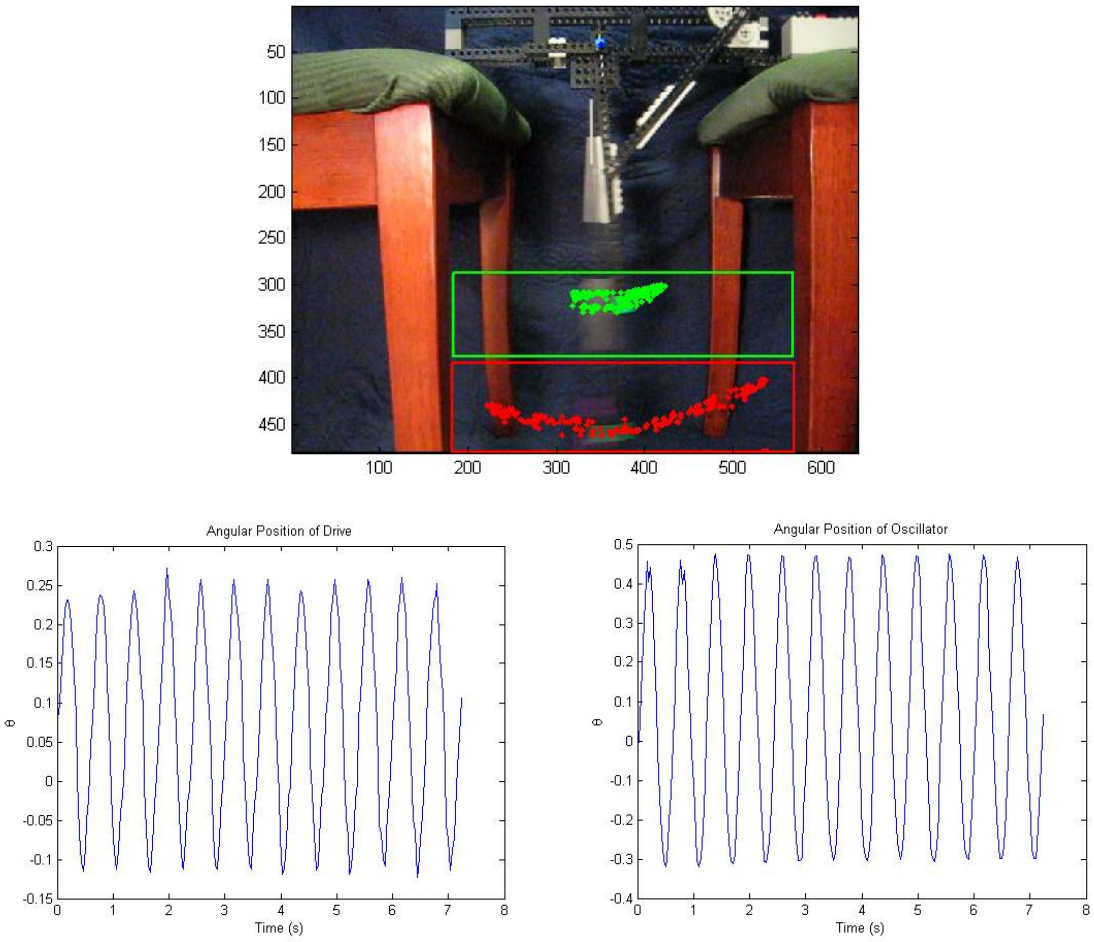


Figure 4: The first frame of a movie (top), with the found locations of the pivot point (blue dot) drive (green dots) and response (red dots) pendulum arms

superimposed. The angle as a function of time for the drive and response arms is shown in the time series on the bottom.

3. Numerical Simulations

Numerical simulations were performed, by numerically solving the differential equation 1 or 2 using NDSolve in Mathematica. All code may be found in Appendix C. Results from Equation 1 will be referred to as exact, while those from Equation 2 will be referred to as approximate.

In order to have somewhat accurate and relevant simulations, the intrinsic parameters of the oscillator first needed to be mapped out. In order to do this, the pendulum was set swinging and its decay was recorded. The data was fit to the standard solution of the damped pendulum;

$$\theta(t) = a\, e^{-\beta t}\, Sin(w\, t\, +\, \varphi) \tag{3}$$

The results of this fit are shown in Figure 5, where a = .439, β = .254, ω = 5.55, and φ = 1.12. These parameters were used in all the simulations that follow.

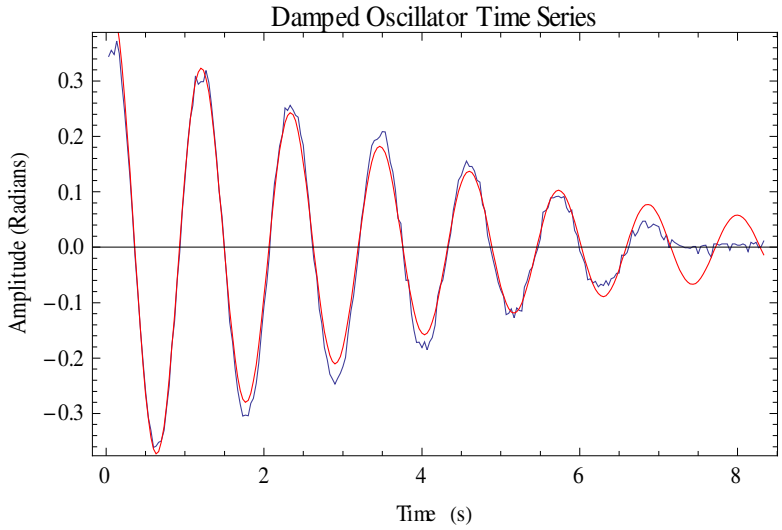


Figure 5: The time series for the non-driven response arm (blue), and fit (red).

Since our system is essentially a driven oscillator, we expect certain behavior, such as resonance. However, how that resonance behaves as we change our parameters (such as D) is not clear, nor is it clear if the behavior of the approximate solution follows that of the exact solution – that is, it is not clear if the approximations we have made are valid. To explore this aspect of the parameter space, simulations were performed on both equations. These were done by numerically solving the equation with initial conditions at rest ($\theta(t = 0) = 0, \dot{\theta}(t = 0) = 0$) for a long time (100 s). The transient behavior was ignored, and the long time behavior (60 to 100 s) was fit using the same algorithm used to fit the data. The driving frequency was then varied over a certain range, and the amplitude of the response vs. the driving frequency was plotted (ie; the resonance curve). The results are shown in Figure 6.

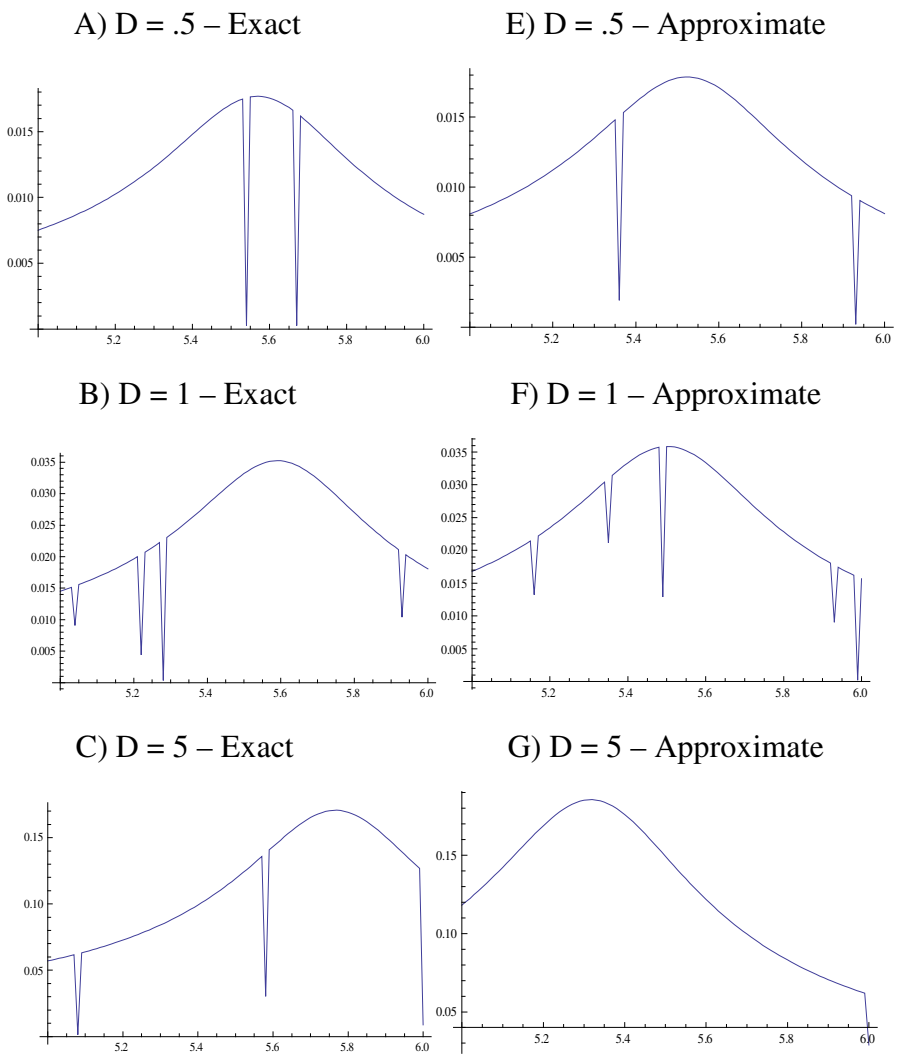


Figure 6: Numerical simulations of the exact (Equation 1) and approximate (Equation 2) resonance curves. The abscissa in each is the frequency of the driving oscillator (radians / s), while the ordinate is the response amplitude. Note that the resonance frequency shifts to higher frequencies for the exact model, while it shifts to lower frequencies for the approximate model. This behavior can be experimentally probed. The periodic spikes downward in amplitude were from a bad fit, and can be ignored.

Interestingly, the predictions of the exact and approximate equations do not agree, while the resonance frequency of the approximate solution decreases as D (the coupling parameter) increases. This result is predictable from Equation 2, for we can re-write; $\omega' = \omega_0^2 - D$ as the new frequency of the oscillator, which clearly decreases with increasing D. Since the resonance frequency is $\omega_{resonance} = \sqrt{\omega'^2 - \beta^2} \sim \omega'$, the resonance peak should shift down. However, the exact solution shows the resonance peak shifting to *higher* frequencies as the coupling strength increases! It is likely that our assumption of small response amplitudes is to blame. If the response amplitude is small, then we can treat it as essentially always following the drive arm, in which case the spring pulls it away from equilibrium – the

opposite way gravity is pulling – hence we change g, the gravitational restoring force of the pendulum to g – D, and since ω ~ g, ω decreases as D increases. However, we are driving a resonance – it would be unusual for the response arm not to exceed the amplitude of the drive arm, and this is exactly the behavior we see, and will be analyzed in the next section.

4. Results

We have already shown that the frequency and damping of the pendulum can be determined, using these, we can calculate the effective length of the pendulum $l = g/w^2 = 31$ cm, very close to the actual length of the pendulum 37 cm from the end to the pivot point. Now, we move on to experimentally determine the resonance curves. Data were taken for four different coupling strengths, and the frequency was changed over the full dynamic range of the motor as described in Section 2 above. The resulting resonance curves are shown below in Figure 7.

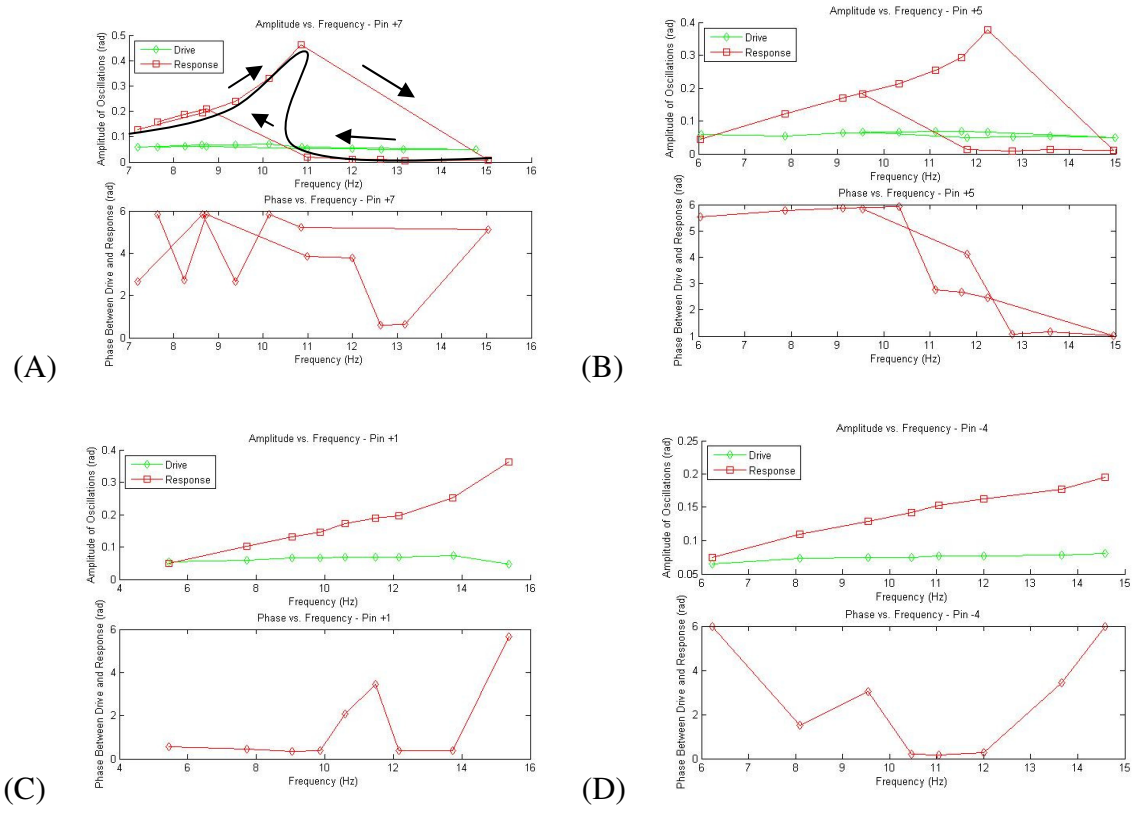


Figure 7: Experimental resonance curves for increasing coupling constant (A-D, the coupling force 'D' (Eq. 1) is indicated by the pin number in the title – higher pin numbers correspond to a lower connection force). Red lines are the amplitude of the response pendulum, while the green lines are the drive oscillator amplitude. The direction of the frequency change (arrows) show a hysteretic effect; increasing frequency (upper line) and decreasing frequency (lower line) are offset, indicative of a nonlinear spring constant. The black curve is an interpolated line drawn to guide the eye to the likely underlying resonance curve structure.

First note the presence of a resonance peak, showing that our driving mechanism is not changing the characteristics of the system too much, also the driving amplitude stays relatively constant, showing that the resonance effect is not purely due to the drive amplitude changing. Although the phase is often noisy, Figure 7B shows the expected phase change of 2π through the resonance. Comparing this experimental data to Figure 6 shows that indeed, the small angle, and small driving amplitude assumption is not valid, as the peak does indeed shift to higher frequencies for increasing coupling strength. Also, note that the peaks are not symmetrical as the simulations predicted. Instead on the increasing frequency side, the amplitude increases to a maximum, then became unstable, and switches to a low frequency oscillation at a high frequency (the speed of the motor greatly increased when the pendulum stopped oscillating at as great an amplitude as the resistance offered by the swinging pendulum disappeared). The pendulum then maintained the low oscillation amplitude as the frequency was decreased until this too became unstable and switched to the higher amplitude (and lower frequency, due to the feedback to the motor) regime. This hysteretic bending of the resonance curve is just what we would expect to see from a nonlinear restoring force, and is consistent with a stiffening spring constant. Unlike the Duffing oscillator, it is likely that this enters through the spring constant in the coupling term going from a linear spring constant to a nonlinear one. Hence the equation of motion for the system should become;

$$\ddot{\theta_r} - 2\beta\dot{\theta_r} - \omega^2 Sin(\theta_r) = 2\ D\ Sin\left(\frac{\theta_d - \theta_r}{2}\right) + k_3\ \left(2\ d\ Sin(\frac{\theta_d - \theta_r}{2})\right)^3 \tag{3}$$

where k_3 is the next order contribution of the spring constant.

5. Discussion and Conclusions

In this paper we have shown that one can quickly prototype a LEGO Duffing oscillator, which behaves much as one would theoretically predict. The parameter space was partially mapped out, as we have shown that the resonance frequency increases as a function of coupling between drive, and response arms. Simulations were run to verify equations, and seem to hold up to rough approximation. They do not capture the hysteretic behavior of the experimental system, likely because they were always started from the same initial conditions (at rest) while in the experimental system the frequency was continuously increased. Additionally any nonlinearity of the spring connecting the two oscillator arms was not taken into account in the simulations, however, the data show this nonlinearity to give rise to the expected bending and unstable equilibriums of oscillation amplitudes expected for a nonlinear oscillator. The LEGO Duffing-like oscillator presented here demonstrates many of the nonlinear dynamics concepts we have learned, in an easy-to-study environment that would be conducive to demonstrations of labs.

```
% Appendix A - MATLAB Code

% A wrapper script that will call the video analysis funciton multiple
% times

for k = 82:94
    fname = strcat('74',num2str(k))
    VideoAnalysisFunction(fname)
end

%-------------------------------------------------------------------------

% Nonlinear Analysis Code
% Kevin Claytor Physics 213 2010-12-10

function VideoAnalysisFunction(fname)

filename = strcat('MVI_',fname,'.avi');
% Load up a movie
vid = mmreader(filename);

nFrames = vid.NumberOfFrames;
frameRate = vid.FrameRate;
% Get the pivot point from the user
frame = read(vid,1);
figure; imagesc(frame);
title(filename);
%[x,y] = ginput(1)
x = 320;
y = 24;
pivx = round(x); pivy = round(y);
hold on;
plot(pivx,pivy,'b.','Markersize',10,'MarkerFaceColor','b')

% Have the user define the region of interest for the red dot
%[roix,roiy] = ginput(2)
roix = [120;526];
roiy = [394;478];
xmin = floor(min(roix));
xmax = ceil(max(roix));
ymin = floor(min(roiy));
ymax = ceil(max(roiy));
plot([xmin,xmax,xmax,xmin,xmin],[ymin,ymin,ymax,ymax,ymin],'r-','LineWidth',2);

% Have the user define the region of interest for the green dot
%[roigx,roigy] = ginput(2)
roigx = [281;400];
roigy = [297;350];
gxmin = floor(min(roigx));
gxmax = ceil(max(roigx));
gymin = floor(min(roigy));
```

```
gymax = ceil(max(roigy));
plot([gxmin,gxmax,gxmax,gxmin,gxmin],[gymin,gymin,gymax,gymax,gymin],'g-','LineWidth',↙
2);

% Initalize our time and angle vectors
time = zeros(1,nFrames);
angle = zeros(1,nFrames);
angleDrive = zeros(1,nFrames);

% Read one frame at a time, analyzing our images for the locations of our
% dots.
for k = 1 : nFrames
    %k = 5;
    updatemsg = sprintf('Analyzing Frame (%d/%d)',k,nFrames);
    disp(updatemsg)
    time(k) = k/frameRate;
    frame = read(vid, k);
    % Take the cut-out ROI of our image, threshold it, and find where the
    % bright red spot is, and record the angle.
    roi_pend = frame(ymin:ymax,xmin:xmax,2);
    roi_pendAdj = imadjust(roi_pend);
    bw = im2bw(roi_pendAdj,.99);
    [~,col] = max(sum(bw,1));
    [~,row] = max(sum(bw,2));
    angle(k) = atan2(xmin+col-pivx,ymin+row-pivy);
    plot(xmin+col,ymin+row,'r.','Markersize',10,'MarkerFaceColor','b')
    % Take the green ROI, threshold it, find where the spot is and record
    % the angle
    roi_drive = frame(gymin:gymax,gxmin:gxmax,2);
    roi_driveAdj = imadjust(roi_drive);
    bw = im2bw(roi_driveAdj,.99);
    [~,gcol] = max(sum(bw,1));
    [~,grow] = max(sum(bw,2));
    angleDrive(k) = atan2(gxmin+gcol-pivx,gymin+grow-pivy);
    plot(gxmin+gcol,gymin+grow,'g.','Markersize',10,'MarkerFaceColor','b')

end

figure;
plot(time,angle);
title('Angular Position of Oscillator');
xlabel('Time (s)');
ylabel('\theta');

figure;
plot(time,angleDrive);
title('Angular Position of Drive');
xlabel('Time (s)');
ylabel('\theta');

% Subtract off any offset to the angle
```

```
angle = angle - mean(angle);
angleDrive = angleDrive - mean(angleDrive);
% Save the output to something that MMA can read:
response_name = strcat(fname,'_response.txt');
drive_name = strcat(fname,'_drive.txt');
csvwrite(response_name,[time',angle']);
csvwrite(drive_name,[time',angleDrive']);

pause(1);
%close all;

return

% For plotting a frame, and the channels
% figure;
% subplot(2,2,1); imagesc(frame); title('RBG image');
% subplot(2,2,2); imagesc(frame(:,:,1)); title('Red channel');
% subplot(2,2,3); imagesc(frame(:,:,2)); title('Green channel');
% subplot(2,2,4); imagesc(frame(:,:,3)); title('Blue channel');
```

Appendix B - Curve Fitting and Parameter Determination

Add the nonlinear directory to our paths, so we can import data easily

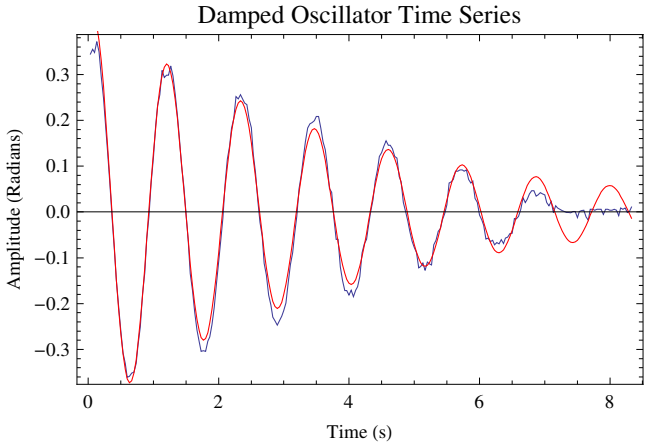
In[11]:=
```
AppendTo[$Path, ToFileName[{$HomeDirectory, "My Documents\Matlab\Nonlinear"}]];
```

■ Damped Oscillation - Determine the intrinsic oscillator properties

In[855]:=
```
data = Import["NoDrive_response.txt", "CSV"];
gd = ListLinePlot[data];

myfit = a * Sin[w t + phi] * Exp[-b t];
fitparams =
 FindFit[data, {myfit, {w > 0, b > 0, 0 ≤ phi ≤ 2 Pi}}, {a, b, w, phi}, t, Method → NMinimize]

newfit = myfit /. fitparams;
time = Transpose[data][[1]];
theory = Transpose[{time, newfit /. t → time}];
gt = ListLinePlot[theory, PlotStyle → Red];

Show[gd, gt, Frame → True, FrameLabel → {"Time (s)", "Amplitude (Radians)"},
 PlotLabel → "Damped Oscillator Time Series"]
```

Out[858]=
```
{a → 0.439071, b → 0.253199, w → 5.55213, phi → 1.12023}
```

Out[863]=



Get the effective length of our pendlum, and the natural frequency:

In[24]:=
```
9.8 / (w /. fitparams)^2
Sqrt[(w^2 + b^2) /. fitparams]
```

Out[24]=
```
0.317912
```

Out[25]=
```
5.5579
```

■ Driven Oscillation - determine drive and response parameters

In[480]:=
```
drivefreq = {};
driveampl = {};
drivephas = {};
respofreq = {};
respoampl = {};
respophas = {};
```

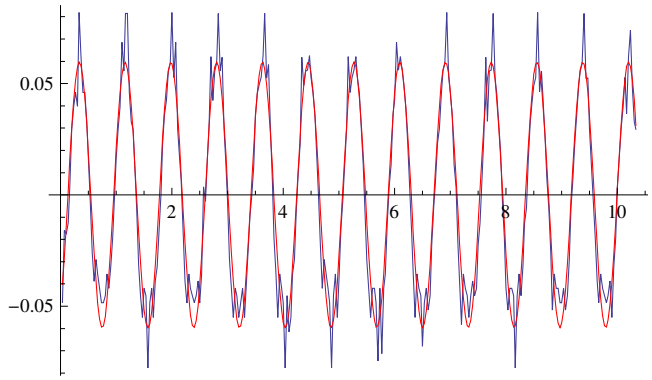
In[820]:=
```
fname = "7494";
drive = Import[fname <> "_drive.txt", "CSV"];
response = Import[fname <> "_response.txt", "CSV"];
time = Transpose[drive][[1]];
angled = Transpose[drive][[2]];
timeo = Transpose[response][[1]];
angleo = Transpose[response][[2]];
(* Plot the data *)
gdd = ListLinePlot[drive];
gdr = ListLinePlot[response];
```

In[829]:=
```
myfit = a * Sin[w t + phi];
fitparamsDrive =
 FindFit[drive, {myfit, {w > 5, 0 < phi < 2 π}}, {a, w, phi}, t, Method → NMinimize]
fitparamsResponse = FindFit[response, {myfit, {w > 5, 0 < phi < 2 π}},
  {a, w, phi}, t, Method → NMinimize]

(* Evaluate and Plot the fit *)
newfitDrive = myfit /. fitparamsDrive;
fitDrive = Transpose[{time, newfitDrive /. t → time}];
newfitResponse = myfit /. fitparamsResponse;
fitResponse = Transpose[{time, newfitResponse /. t → time}];
gtd = ListLinePlot[fitDrive, PlotStyle → Red];
gtr = ListLinePlot[fitResponse, PlotStyle → Red];

Show[gdd, gtd]
Show[gdr, gtr]
```

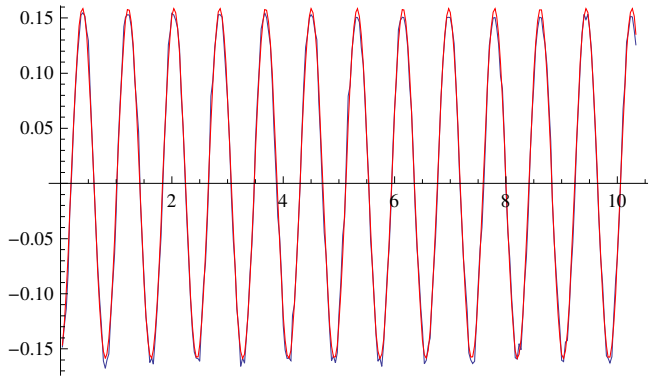
Out[830]= {a → 0.0595959, w → 7.64416, phi → 5.28618}

Out[831]= {a → 0.158672, w → 7.64045, phi → 4.85484}

Out[838]=



Out[839]=



In[840]:=
```
AppendTo[drivefreq, Abs[w /. fitparamsDrive]]
AppendTo[driveampl, Abs[a /. fitparamsDrive]]
AppendTo[drivephas, Abs[phi /. fitparamsDrive]]
AppendTo[respofreq, Abs[w /. fitparamsResponse]]
AppendTo[respoampl, Abs[a /. fitparamsResponse]]
AppendTo[respophas, Abs[phi /. fitparamsResponse]]
```

Out[840]= {7.2026, 8.63769, 9.37891, 10.1373, 10.8649, 14.7588, 13.131, 12.6435, 11.9923, 10.9865, 8.73106, 8.25077, 7.64416}

Out[841]= {0.0593978, 0.0687242, 0.068968, 0.0712477, 0.0600876, 0.0510611, 0.0509974, 0.0503264, 0.0513835, 0.0524363, 0.0604971, 0.062686, 0.0595959}

Out[842]= {4.67662, 1.47521, 5.89395, 5.27203, 1.2321, 1.15666, 5.16264, 5.67536, 1.11421, 4.61548, 0.589975, 0.0591598, 5.28618}

Out[843]= {7.19904, 8.64038, 9.38299, 10.1343, 10.8564, 15.0312, 13.1702, 12.6218, 11.9908, 10.9794, 8.73167, 8.24658, 7.64045}

Out[844]= {0.127839, 0.193401, 0.240783, 0.32937, 0.462744, 0.00628834, 0.00583246, 0.0116338, 0.011534, 0.0209298, 0.208276, 0.187116, 0.158672}

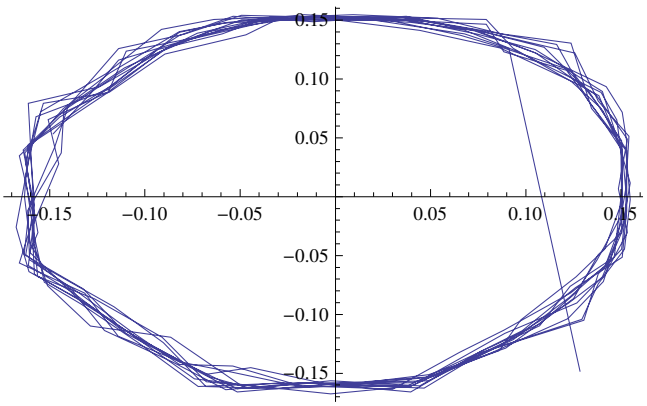
Out[845]= {1.04018, 1.01353, 2.2507, 4.81887, 0.182646, 6.28319, 5.79737, 0., 4.90401, 2.18127, 0.149943, 2.77488, 4.85484}

■ Time delay plotting

In[917]:=
```
offset = 6
velo = RotateLeft[angleo, offset];
angleo2 = angleo[[ ;; -offset]];
velo2 = velo[[ ;; -offset]];
ListLinePlot[Transpose[{angleo2, velo2}]]
```

Out[917]= 6

Out[921]=



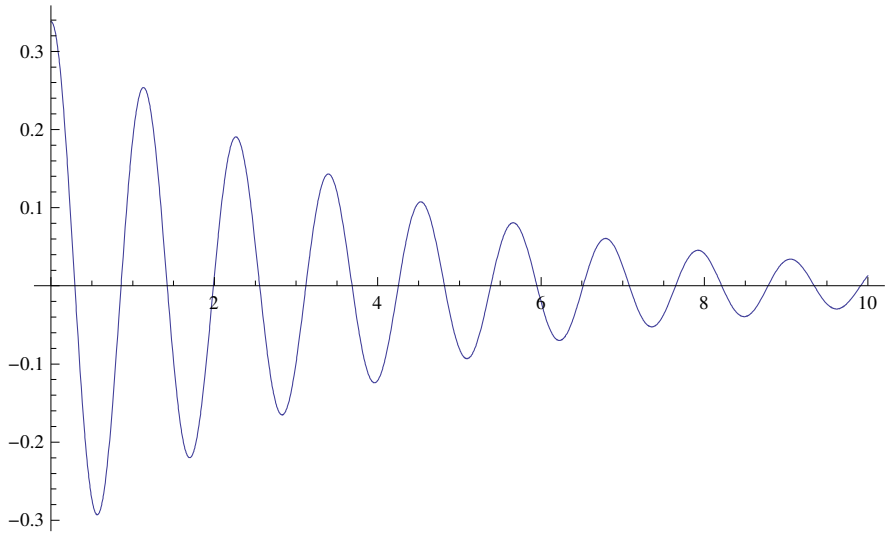
Appendix C - Numerical Simulations

■ Set up the system with the same physical parameters

In[864]:=
```
b = .2532;
w0 = 5.558;
tmax = 10;
soln =
 NDSolve[{x''[t] + 2 b x'[t] + w0^2 x[t] == 0, x[0] == .338, x'[0] == 0}, {x[t]}, {t, 0, tmax}]
Plot[Evaluate[x[t] /. soln], {t, 0, tmax}, PlotRange → All]
```

Out[867]=
```
{{x[t] → InterpolatingFunction[{{0., 10.}}, <>][t]}}
```

Out[868]=



■ Numerical Simulations with varying coupling strength

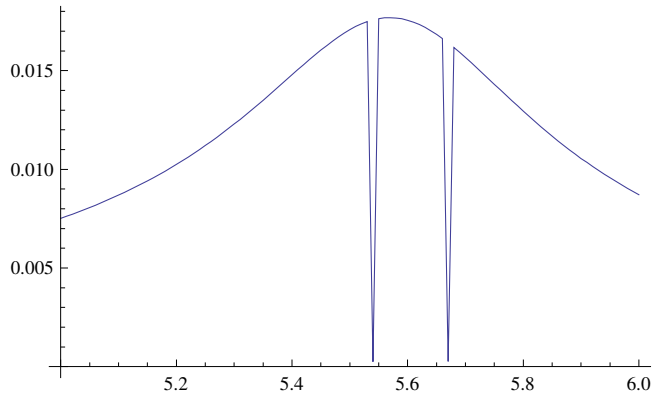
Set up the general ploting parameters

In[869]:=
```
tmin = 60;
tmax = 100;
tstep = .05;
time = Table[t, {t, tmin, tmax, tstep}];

amp = .2;
```

■ d=.5

In[874]:=
```
(* The our variable parameter *)
d = .5;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + w0^2 x[t] == d Sin[(amp Sin[w2 t] - x[t]) / 2],
    x[0] == 0, x'[0] == 0}, {x[t]}, {t, 0, tmax}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

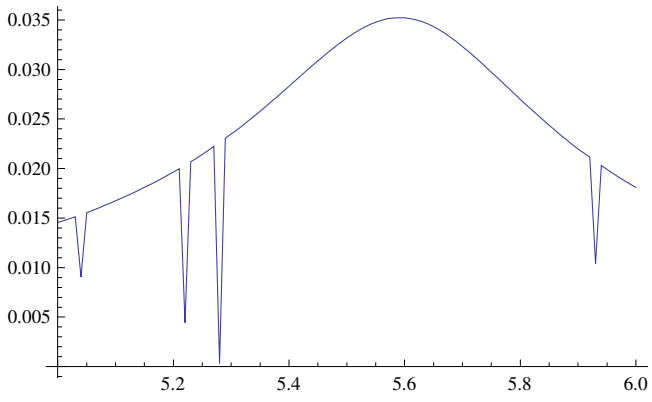
Out[878]=



d=1

In[879]:=
```
(* The our variable parameter *)
d = 1;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + w0^2 x[t] == d Sin[(amp Sin[w2 t] - x[t]) / 2],
    x[0] == 0, x'[0] == 0}, {x[t]}, {t, 0, tmax}];
 time = Table[t, {t, tmin, tmax, tstep}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

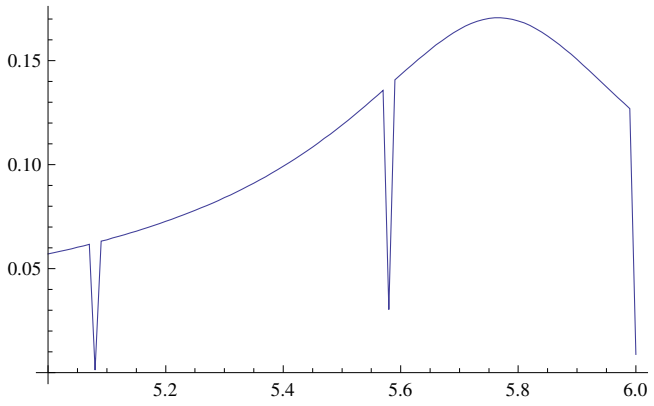
Out[883]=



- **d=5**

In[884]:=
```
(* The our variable parameter *)
d = 5;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + w0^2 x[t] == d Sin[(amp Sin[w2 t] - x[t]) / 2],
    x[0] == 0, x'[0] == 0}, {x[t]}, {t, 0, tmax}];
 time = Table[t, {t, tmin, tmax, tstep}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

Out[888]=



- **Numerical Simulations with Small Angle Approximation - varying d - coupling strength**

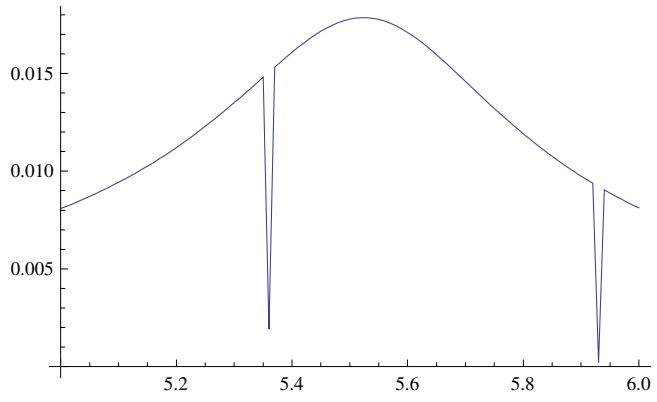
Set up the general ploting parameters

```
tmin = 60;
tmax = 100;
tstep = .05;

amp = .2;
```

d=.5

In[889]:=
```
(* The our variable parameter *)
d = .5;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + (w0^2 - d / 2) x[t] == d amp Sin[w2 t] / 2, x[0] == 0, x'[0] == 0},
   {x[t]}, {t, 0, tmax}];
 time = Table[t, {t, tmin, tmax, tstep}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

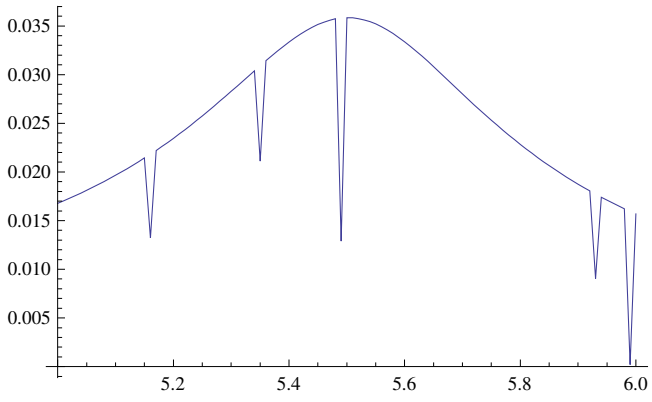
Out[893]=



■ d=1

In[894]:=
```
(* The our variable parameter *)
d = 1;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + (w0^2 - d / 2) x[t] == d amp Sin[w2 t] / 2, x[0] == 0, x'[0] == 0},
   {x[t]}, {t, 0, tmax}];
 time = Table[t, {t, tmin, tmax, tstep}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

Out[898]=



■ d=5

In[899]:=
```
(* The our variable parameter *)
d = 5;

wvec = {};
ampvec = {};
Do[
 soln = NDSolve[{x''[t] + 2 b x'[t] + (w0^2 - d / 2) x[t] == d amp Sin[w2 t] / 2, x[0] == 0, x'[0] == 0},
   {x[t]}, {t, 0, tmax}];
 time = Table[t, {t, tmin, tmax, tstep}];
 ndata = Flatten[Table[Evaluate[x[t] /. soln], {t, tmin, tmax, tstep}]];
 ntdata = Transpose[{time, ndata}];
 gnd = ListLinePlot[ntdata];

 myfit = a * Sin[w t + phi];
 fitparams = FindFit[ntdata, {myfit}, {a, w, phi}, t, Method → NMinimize];
 newfit = myfit /. fitparams;
 fitfinal = Transpose[{time, newfit /. t → time}];
 gtd = ListLinePlot[fitfinal, PlotStyle → Red];

 Show[gnd, gtd];

 AppendTo[wvec, w2];
 AppendTo[ampvec, Abs[a /. fitparams]];
 , {w2, 5, 6, .01}]

ListLinePlot[Transpose[{wvec, ampvec}], PlotRange → All]
```

Out[903]=