Lab Report 3

Team Members: Joie Green, Melissa Prado, Niva Razin, Alexander Ross

EA4 Lab Certification

We certify that the members of the team named below worked on this lab using only our own ideas, possibly with help from the TAs and our professors, specifically in the writing of the report, the programming and the analysis required for each task. We did not collaborate with any members of any other team nor did we discuss this lab or ask assistance from anyone outside of our team, the TAs and our professors. If we accessed any websites in working on these labs, these websites are explicitly listed in our report (with the exception of websites used only to get information on specific Matlab programs). We understand that there will be severe consequences if this certification is violated and there is unauthorized collaboration.

1. Name NIVA RAZIN Signature [illegible] Date 2/14/2020
2. Name Melissa Prado Signature [illegible] Date 2/14/2020
3. Name Joie Green Signature [illegible] Date 2/4/2020
4. Name Alexander Ross Signature [illegible] Date 2/14/2020

INTRODUCTION

In this lab, we examine the behavior of linear and nonlinear oscillators. We derive mathematical models that progress our intuition and explain the behavior of these systems. We apply the Improved Euler (IE) method to model nonlinear oscillations of a Duffing oscillator by modifying the code from our previous population modeling lab.We developed a differential equation solver in MATLAB based on the IE method and implemented it to better understand and model the behavior of the Duffing oscillator. In turn, we gained valuable experience in MATLAB coding, software development, mathematical modeling, critical thinking/analysis, writing, collaboration, and project management.

TASK 1

Given the unforced, undamped oscillator described by the equation:

$$mx'' + kx + \beta x^3 = 0 \tag{1}$$

with initial conditions $x(0) = A$, $x'(0) = 0$, we set the mass $m = 1$ and spring constant $k = 1$. These values give the equation:

$$x'' + x + \beta x^3 = 0\,. \tag{2}$$

We determine an equation for the total mechanical energy of the system by building off of the equation:

$$E_{total} = KE + PE\ , \tag{3}$$

or the total mechanical energy is equal to the sum of the kinetic and potential energies of the system, KE and PE, respectively. KE is given by:

$$KE = \tfrac{1}{2}mv^2\,, \tag{4}$$

with mass m and velocity v. Given that $m = 1$ in this system and that velocity is the derivative of displacement x, we can rewrite (4) as:

$$KE = \tfrac{1}{2}(1)(x')^2 = \tfrac{1}{2}x'^2\,.$$

Potential energy can be evaluated of as the opposite of work, given by:

$$PE = \ -W = \ -\int F(x)\, dx\,.$$

$F(x) = mx'' = -kx - \beta x^3$ by Newton's second law of motion and observation of (1):

$$PE = -\int(-kx - \beta x^3)\,dx = \tfrac{1}{2}kx^2 + \tfrac{1}{4}\beta x^4 = \tfrac{1}{2}x^2 + \tfrac{1}{4}\beta x^4$$

when $k = 1$. We plug these expressions for kinetic and potential energy into (4):

$$E = \tfrac{1}{2}x'^2 + \tfrac{1}{2}x^2 + \beta\tfrac{1}{4}x^4\,. \tag{5}$$

We see that E is constant for all initial conditions by taking its derivative with respect to time:

$$\tfrac{dE}{dt} = \tfrac{dx(t)}{dt} = x'\,x'' + x\,x' + \beta x^3 x' = x'\,(x'' + x + \beta x^3) = 0\,.$$

Factoring x' out of the derivative, we get (2), which is equal to zero (constant).

TASK 2

We eliminate the nonlinear behavior of the oscillator by setting $\beta = 0$:

$$mx'' + kx = 0\,. \tag{6}$$

We solve the exact solution to (6) using the initial conditions $x(0) = A$ and $x'(0) = 0$. We begin by setting (6) in terms of the characteristic polynomial with $m = 1$, $k = 1$, and $\omega_o^2 = k/m = 1$:

$$r^2 + \omega_o^2 = 0\,.$$

We solve for the roots:

$$\sqrt{r^2} = r = \pm\sqrt{-\omega_o^2} = \pm\,\omega_o i\,.$$

These imaginary roots indicate a particular solution of the form:

$$e^{ax}(c_1 cos(bx) + c_2 sin(bx))\,.$$

with complex conjugate roots $a \pm bi$. We plug in for $a = 0$ and $b = \omega_o$:

$$x(t) = c_1 cos(\omega_o x) + c_2 sin(\omega_o x)\,.$$

We use the initial conditions to solve for c_1 and c_2:

$$x(0) = A = c_1 cos(0) + c_2 sin(0) \rightarrow c_1 = A$$

$$x(t) = Acos(\omega_o x) + c_2 sin(\omega_o x)$$

$$x'(t) = -A\omega_o sin(\omega_o x) + c_2\omega_o cos(\omega_o x)$$

$$x'(0) = 0 = \; -A\omega_o sin(0) + c_2\omega_o cos(0) = c_2\omega_o = 0 \rightarrow c_2 = 0\,.$$

Thus, the particular solution is:

$$x(t) = Acos(\omega_o x)$$

with amplitude A and a period of motion T:

$$T = \tfrac{2\pi}{\omega_o} = 2\pi\,.$$

Now, we plug the initial conditions and the parameter $\beta = 0$ into (5) to express E in terms of A:

$$E = \tfrac{1}{2}(0)^2 + \tfrac{1}{2}A^2 + (0)\tfrac{1}{4}x^4 = \tfrac{1}{2}A^2\,.$$

Again taking $\beta = 0$, we solve for x' in (5):

$$E = \tfrac{1}{2}x'^2 + \tfrac{1}{2}x^2 + (0)\tfrac{1}{4}x^4 = \tfrac{1}{2}x'^2 + \tfrac{1}{2}x^2$$

$$2E = x'^2 + x^2 \rightarrow 2E - x^2 = x'^2$$

$$x'(t) = \tfrac{dx}{dt} \; = \; \pm\sqrt{2E - x^2}\,.$$

We then solve the separable equation for the period T, which has units of time:

$$\frac{dx}{\sqrt{2E-x^2}} = dt$$

$$\int \frac{dx}{\sqrt{2E-x^2}} = \int dt \; = T$$

We substitute $E = \tfrac{1}{2}A^2$ and evaluate the integral with respect to x on the interval from 0 to A. These integral bounds give only a quarter of the full period, so we also multiply by four:

$$T = 4\int_0^A \frac{dx}{\sqrt{2(\frac{1}{2}A^2)-x^2}} = 4\int_0^A \frac{dx}{\sqrt{A^2-x^2}} = 4\left[sin^{-1}(\tfrac{x}{A})\right]_0^A = 4\left[sin^{-1}(\tfrac{A}{A}) - sin^{-1}(\tfrac{0}{A})\right] = 4sin^{-1}(1) = 2\pi$$

Thus, the period for all of A is $T = 2\pi$.

TASK 3

We again plug the initial conditions into (5) to express E in terms of A:

$$E = \tfrac{1}{2}(0)^2 + \tfrac{1}{2}(A)^2 + \tfrac{1}{4}\beta(A)^4 = \tfrac{1}{2}A^2 + \tfrac{1}{4}\beta(A)^4 = \tfrac{1}{2}A^2 + \tfrac{1}{4}\beta A^4$$

Now taking $\beta > 0$, we again solve for x' in (5):

$$-\tfrac{1}{2}x^2 - \beta\tfrac{1}{4}x^4 + E = \tfrac{1}{2}x'^2 \quad \rightarrow \quad x'^2 = 2E - x^2 - \beta\tfrac{1}{2}x^4$$

$$x' = \tfrac{dx}{dt} = \sqrt{2E - x^2 - \beta\tfrac{1}{2}x^4}$$

We solve the separable equation, plugging in the new expression for E and following the logic of Task 2 for boundary limits (0 to A), period, and amplitude:

$$\frac{dx}{\sqrt{2E-x^2-\beta\frac{1}{2}x^4}} = dt \quad \rightarrow \int \frac{dx}{\sqrt{2E-x^2-\beta\frac{1}{2}x^4}} = \int dt = T$$

$$T = 4\int_0^A \frac{dx}{\sqrt{2E-x^2-\beta\frac{1}{2}x^4}} = 4\int_0^A \frac{dx}{\sqrt{2\left(\frac{1}{2}A^2+\frac{1}{4}\beta A^4\right)-x^2-\beta\frac{1}{2}x^4}} = 4\int_0^A \frac{dx}{\sqrt{A^2+\frac{1}{2}\beta A^4-x^2-\beta\frac{1}{2}x^4}}$$

We then make the substitution $x = Ay$ in the integrand and change the bounds accordingly with $y = \frac{x}{A}$:

$$\frac{T}{4} = \int_0^{\left(\frac{A}{A}\right)} \frac{d(Ay)}{\sqrt{A^2+\frac{1}{2}\beta A^4-(Ay)^2-\beta\frac{1}{2}(Ay)^4}} = \int_0^1 \frac{Ady}{\sqrt{A^2+\frac{1}{2}\beta A^4-A^2y^2-\beta\frac{1}{2}A^4y^4}}$$

We factor A^2 out of the root in the denominator and simplify:

$$\frac{T}{4} = \int_0^1 \frac{Ady}{\sqrt{A^2\left(1+\frac{1}{2}\beta A^2-y^2-\beta\frac{1}{2}A^2y^4\right)}} = \int_0^1 \frac{Ady}{A\sqrt{1+\frac{1}{2}\beta A^2-y^2-\beta\frac{1}{2}A^2y^4}} = \int_0^1 \frac{dy}{\sqrt{1+\frac{1}{2}\beta A^2-y^2-\beta\frac{1}{2}A^2y^4}} = \int_0^1 \frac{dy}{\sqrt{1-y^2+\frac{1}{2}\beta A^2(1-y^4)}}.$$

We take $\gamma = \beta A^2$:

$$T = 4\int_0^1 \frac{dy}{\sqrt{1-y^2+\frac{1}{2}\gamma(1-y^4)}},$$

In turn, we see that the period T depends exclusively on $\gamma = \beta A^2$, as y is a dummy variable, only taking on values of one and zero. We also see that T decreases as γ increases; as γ approaches infinity, the integrand approaches zero, as does T. We see T approaches 2π by taking the limit of the integral as γ approaches zero:

$$T(\gamma=0) \approx 4\lim_{\gamma\to 0}\int_0^1 \frac{1}{\sqrt{1-y^2+\frac{1}{2}\gamma(1-y^4)}} = 4\int_0^1 \frac{dy}{\sqrt{1-y^2}} = 4\left[sin^{-1}(y)\right]_0^1 = 4\left[sin^{-1}(1)-sin^{-1}(0)\right] = 4sin^{-1}(1) = 2\pi$$

This model follows our engineering intuition in several ways. Initial displacement A describes how much the spring is initially stretched or compressed from its rest position. By Hooke's law, the restorative force of the spring (i.e., the force pulling/pushing the spring back to its rest position) has a linear relationship with this displacement *within a certain range*. That is, the amount of initial displacement of the spring is directly proportional to the force with which the spring 'bounces back'. Thus, we reason that the period T approaches that of a linear oscillator when the displacement nears or is within this range governed by Hooke's law (i.e., as initial displacement approaches zero). Once the spring is stretched or compressed beyond that range, it will no longer bounce back in a linear way, as its velocity is no longer constant. The coefficient β describes the magnitude of the stretch/compression beyond the range and the magnitude of this nonlinear behavior. The more the spring is stretched or compressed, the greater the restorative force is for a hard spring and the faster the spring bounces back to its rest position. That is, greater initial displacement and nonlinear behavior result in a stronger restorative force, requiring less time for the spring to bounce back, or a smaller T. In the case of a soft spring where the restorative force becomes *weaker* due to nonlinear behavior, displacement has the opposite effect; it takes longer for the spring to bounce back (larger T) when the spring is stretched/compressed more.

TASK 4

Task 4 asks us to consider the unforced, damped oscillator described by the equation:

$$mx'' + kx' + \beta x^3 = 0 \tag{1}$$

We are first asked to write a first order system for $x(t)$ and $y(t) = x'(t)$. This can be done easily by rearranging equation (1) such that the x' is isolated on the left hand side:

$$y(t) = x'(t)$$

$$y(t) = -\tfrac{k}{m}(x) - \tfrac{\beta}{m}(x^3)$$

Subsequently, we modified the programs adapt.m and prime.m to solve computationally analyze our unforced, damped oscillator equation. The code was structured in the following way: The adapt.m program first goes through a while loop meant to determine if the timestep *h* is sufficiently small; if the timestep is too large, the program divides *h* by a set value of *hcut* in order to lower the value. Once the timestep is sufficiently small, the program then runs a series of if statements to determine how the improved euler method is to be implemented. Ordinarily, the improved euler method is implemented, however when the the solution is crossing zero, an alternate routine of linear interpolation is run to determine the slope at those points. This is to get a better approximate of the slope at these points and to pinpoint the zero crossings. Adapt.m also calls another program called prime.m which simply computes dx and dy for the given differential equation. Finally, the program returns several values which are tabulated below, including the count, the average period, and the error of the period.

From this analysis we were able to estimate the average period of the oscillator, as well as the approximate error of the computational work done. The results of this analysis can be seen below in *table 1* and *table 2:*

ε	*errper*	*count*	*raccept*	*hcut*
.1	7.55800×10^{-1}	186	0.86	1.5
.01	4.30700×10^{-1}	556	0.87	1.5
.001	1.69400×10^{-1}	1750	0.88	1.5
.0001	9.42000×10^{-2}	5506	0.89	1.5
.00001	1.14000×10^{-2}	17354	0.89	1.5
.000001	8.70000×10^{-3}	54834	0.89	1.5
.0000001	8.95920×10^{-4}	188760	0.93	2

Table 1: *Validation results for Adapt.m.*

ε_{low}	ε_{high}	$\frac{Count_{high}}{Count_{low}}$	$\sqrt{10}$	$\frac{Errper_{low}}{Errper_{high}}$
.1	.01	2.99	3.162	1.75
.01	.001	3.15	3.162	2.54
.001	.0001	3.15	3.162	1.80
.0001	.00001	3.15	3.162	8.26
.00001	.000001	3.16	3.162	1.31
.000001	.0000001	3.16	3.162	4.14

Table 2: *Relationship between computational work and error for different values of epsilon with hcut = 1.5*

Upon review of *table 1* and *table 2*, we can see that several trends emerge from our computations. First, we notice that as **ε** decreases, *errper* and *count* increase. This shows that the computational error increases when the computational work done is greater. Furthermore, we notice that this relationship between computational error and computational work is non-linear, because from *table 2*, we see that the quotient of the *count* and the quotient of the *errper* are not proportional to one another, which is supported by the non-linear equation we found previously. We also notice from *table 1* that *raccept* is not particularly sensitive to changes in **ε**. This is indicative that the adaptive procedure we implemented is working correctly, as it suggests that the number of steps increases proportional to the count. A stable *raccept* across a wide range of epsilon values signifies that our computational error is consistent.

We recall from previously in the course that for a second order system, to get a solution ten times better than your previous solution, you need to decrease 'h' by a factor of $\sqrt{10}$. This is not directly applicable to adaptive procedures however, because the value of 'h' adapts depending on the input data given and the necessary tolerance of the solution. However, in this case, it is valid due to the role of *count* in our adaptive procedure. *Count*, which is a measure of the computational work in our program, scales directly with the quality of our solution. So, even though 'h' as a measure of computational work is not scaled, the *count* is scaled, and is done so fairly proportionally as can be seen in *table 2*. The $\frac{Count_{high}}{Count_{low}}$ is fairly consistent across different values of epsilon, and moreover, changes by almost exactly a factor of $\sqrt{10}$, thus supporting the conclusion that the second order system model is valid in this case.

Lastly, we see that in *table 1*, an hcut value of 2 gives a better solution than an hcut value of 1.5, however this solution is more costly. This is because *raccept* and *count* are increased significantly with an hcut value of 2, meaning that the amount of computational work needed to get the solution is much greater. This difference is quite significant in this case, as the amount of time needed for the program to compute the period was much greater with an hcut of 2. This is supported by the drastic increase in *count* from an hcut value of 2 to an hcut value of 1.

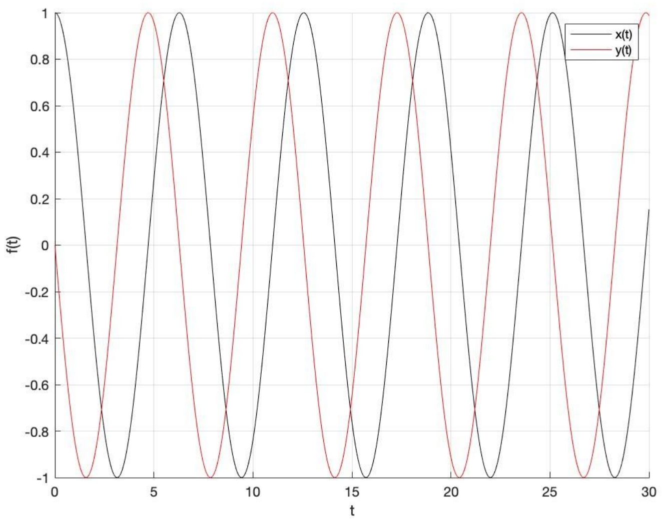

Figure 1: *Plot of x(t) and y(t) for* $\boldsymbol{\varepsilon}$ *= 0.01*

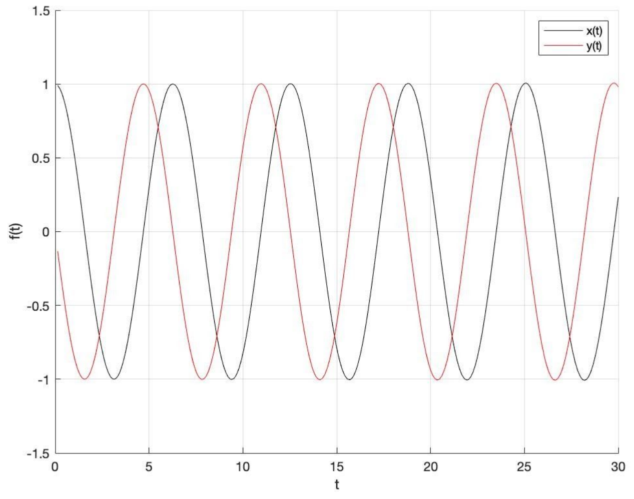

Figure 2: *Plot of x(t) and y(t) for **ε** = 0.0000001*

As can be seen above, **Figures 1** and **2** depict x(t) as a cosine wave and y(t) as a negative sine wave for two different values of epsilon. This is reasonable given the exact solutions of x(t) and y(t), because as we showed previously, y(t) is simply the derivative of x(t). The derivative of cos(x) is -sin(x), which is why y(t) appears as the function -sin(x). This also corresponds to a phase shift of 90 degrees on a plot of x(t) and y(t), which is why **Figures 1** and **2** show y(t) as a phase-shifted sinusoid of x(t).

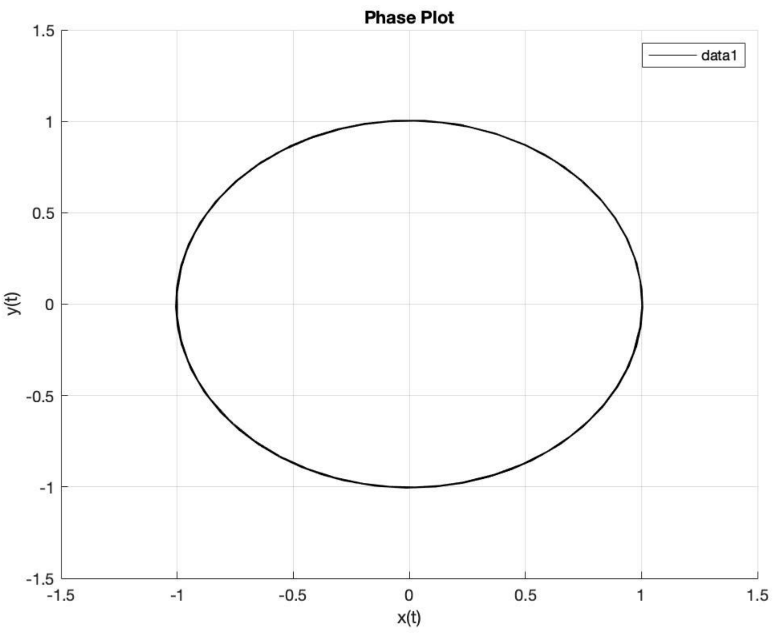

Figure 3: *Phase plot of x(t) vs y(t) for **ε** = 0.01*

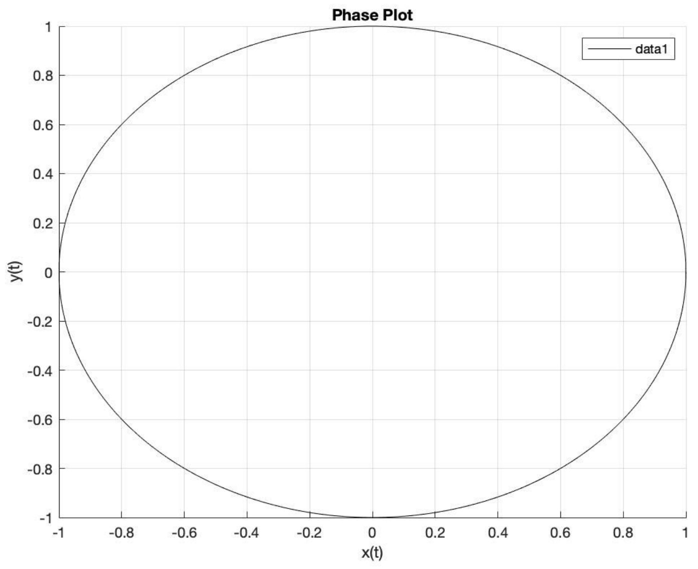

Figure 4: *Phase plot of x(t) vs y(t) for* **ε** *= 0.0000001*

As can be seen above, **Figures 3** and **4** depict phase plots of x(t) versus y(t) for two different values of epsilon. Each of the graphs appear as circles of radius A. This is a reasonable result given the exact solution to the differential equation, as we know that y(t) is the derivative of x(t). Knowing that x(t) and y(t) are sinusoidal phase shifts of one another, we can deduce their phase plot will appear as a circle, as the relationship between their values is perfectly cyclic.

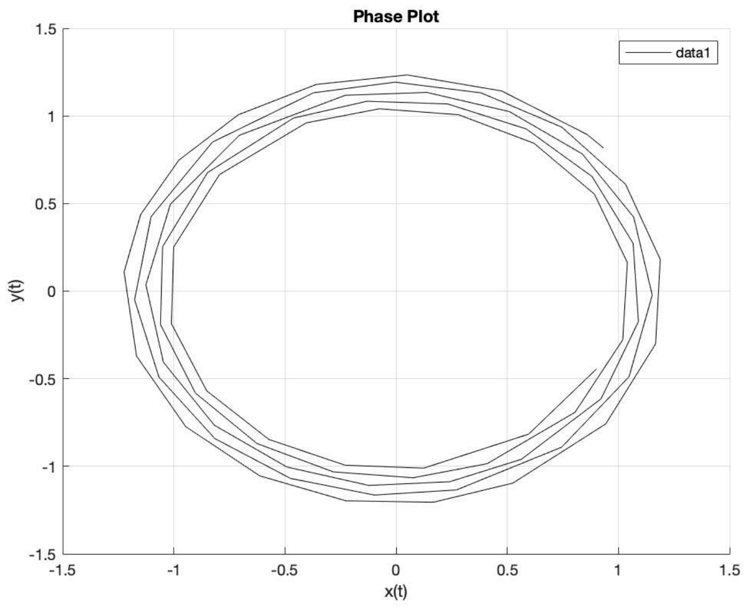

Figure 5: *Phase plot of x(t) and y(t) for* **ε** *= 0.1*

As can be seen above, **Figure 5** depicts a phase plot of x(t) versus y(t) for epsilon = 0.1. We can clearly reject the results of this approximation, as the circle that is produced for this value of epsilon is nowhere near as exact as that of **Figures 3** and **4**. Comparatively, the next highest value of epsilon (epsilon = 0.01) shown in **Figures 3** is much more exact, as the circle produced is much more smooth than in **Figure 5**. This is because the amount of computational work done is greater for lower values of epsilon, resulting in a more accurate phase plot.

<u>TASK 5</u>

In this task, the program *adapt.m* is modified so that it computes the period for an array of A. The modified program is called *adaptperiod.m.* All computations in this task use $\varepsilon = 0.00001$, as well as the same parameters as in Task 4 except β and tfin.

Because errors can no longer be computed for the period since β is now nonzero, two approximations are used:

$$T_2(\gamma) = 2\pi(1 - \tfrac{3}{8}\gamma) \tag{7}$$

$$T_3(\gamma) = 2\pi(1 - \tfrac{3}{8}\gamma + \tfrac{57}{265}\gamma^2) \tag{8}$$

The period is calculated using *adaptperiod.m* for A = 0.1 to A=3 in increments of 0.1, resulting in 30 periods. The calculated period is then plotted against the two approximations, shown in **Figure 6** below.

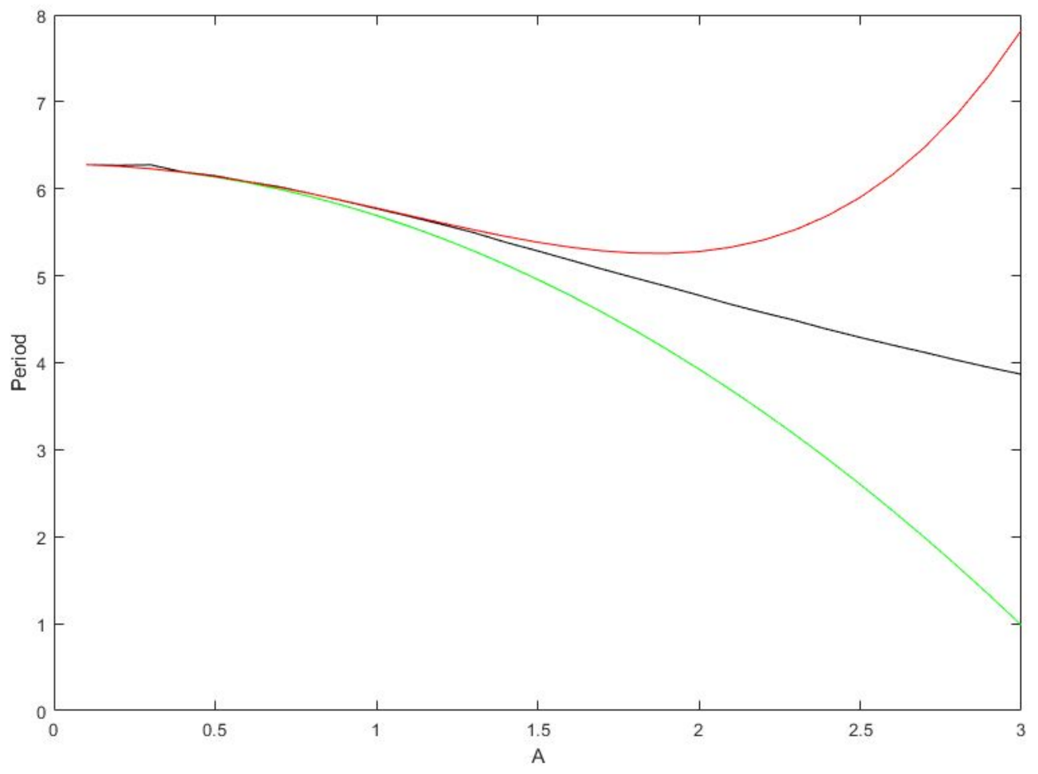

Figure 6: *Plot of two approximations of the period for* $\beta = 0.25$ *(T_2 in green and T_3 in red) along with the computed period T(A).*

As we see above, the approximation T_2 is valid up to about A=0.6, at which point it begins to underestimate the period. The approximation T_3 begins to deviate from the calculated period at about A=1.2. The agreement between the two approximations for small values of A further validates the accuracy of the program.

The solution is then illustrated by plotting x(t) and y(t) against time on the same graph for A=3 (**Figure 7a**).

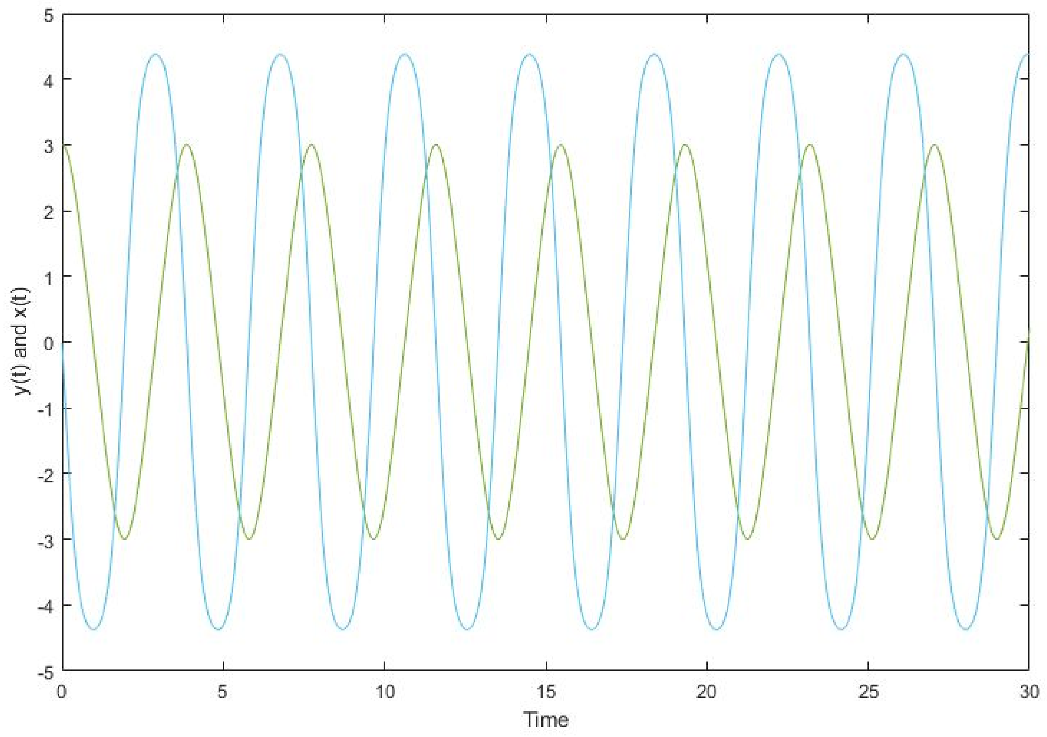

Figure 7a: *Plot of y(t) and x(t) against time for* $\beta = 0.25$.

In order to better illustrate the solution, a phase plot for A =1, A=2, and A=3 is also created (**Figure 7b**).

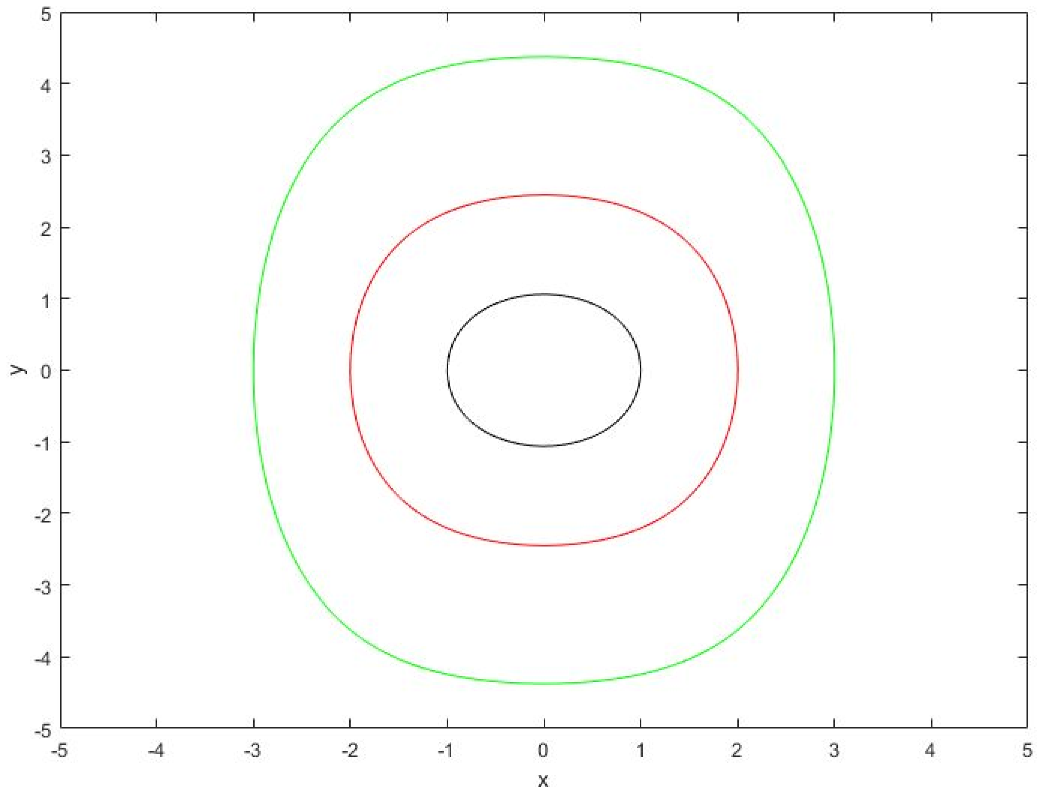

Figure 7b: *Phase plot of x(t) and y(t) for $\beta = 0.25$. Green is A=3, red is A=2, and black is A=1.*

The phase plot is more revealing than **Figure 7a** because it better illustrates how changing β from zero to a nonzero value affects the amplitude, which is the radius of the circles when β is zero, of y but not x. This results in an oval with a radius of A in the x-axis and a larger radius in the y axis.

We then set $\beta = -0.25$ and compute the periods for A=.1 to A=1.9 in 0.1 increments. Tfin is increased to 50 in order to accommodate the increasing periods. We again plot the computed periods against the approximations T_2 and T_3 in **Figure 8**.

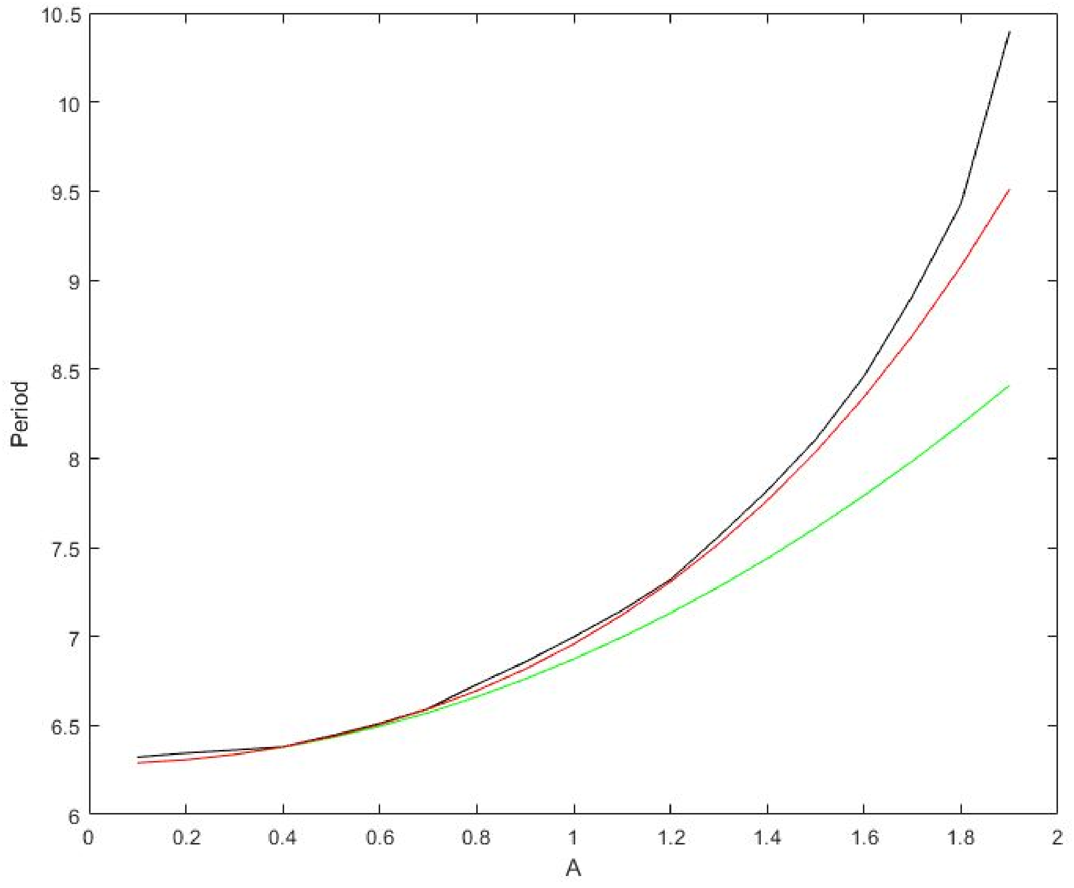

Figure 8: *Plot of two approximations of the period for* $\beta = 0.25$ *(T_2 in green and T_3 in red) along with the computed period T(A).*

Both approximations are valid up to about A=0.7, at which point they both begin to underestimate the period. However, T_3 seems to underestimate the period only slightly up until A=1.2, where it more definitively deviates from the computed period.

The solution x(t) and y(t) are plotted together against time t in **Figure 9a** for A=1.9.

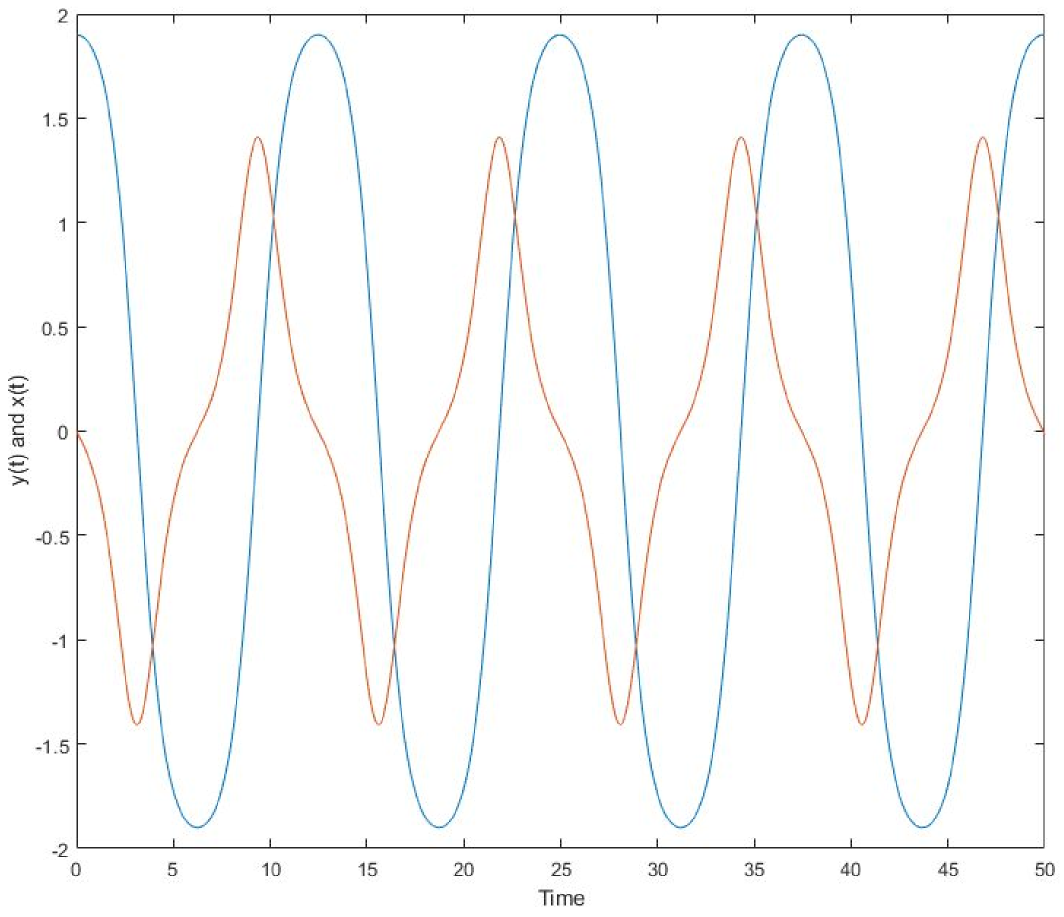

Figure 9a: *Plot of y(t) and x(t) against time for* $\beta = -0.25$.

The x(t) curves for $\beta < 0$ are generally similar to those of $\beta > 0$ and $\beta = 0$. The y(t) curves, however, are vastly different from the other cases. The y(t) curves have a "cinched" appearance to them and also have a smaller amplitude than the x(t) curves, whereas in the case of $\beta > 0$ the y(t) curves had the greater amplitude.

A is then set to 2.1 and tfin to 2.5 and is then plotted on a phase plot. This is then plotted along with A=1.6, A=1.7, A=1.8, and A=1.9 using tfin=50 (**Figure 9b**).

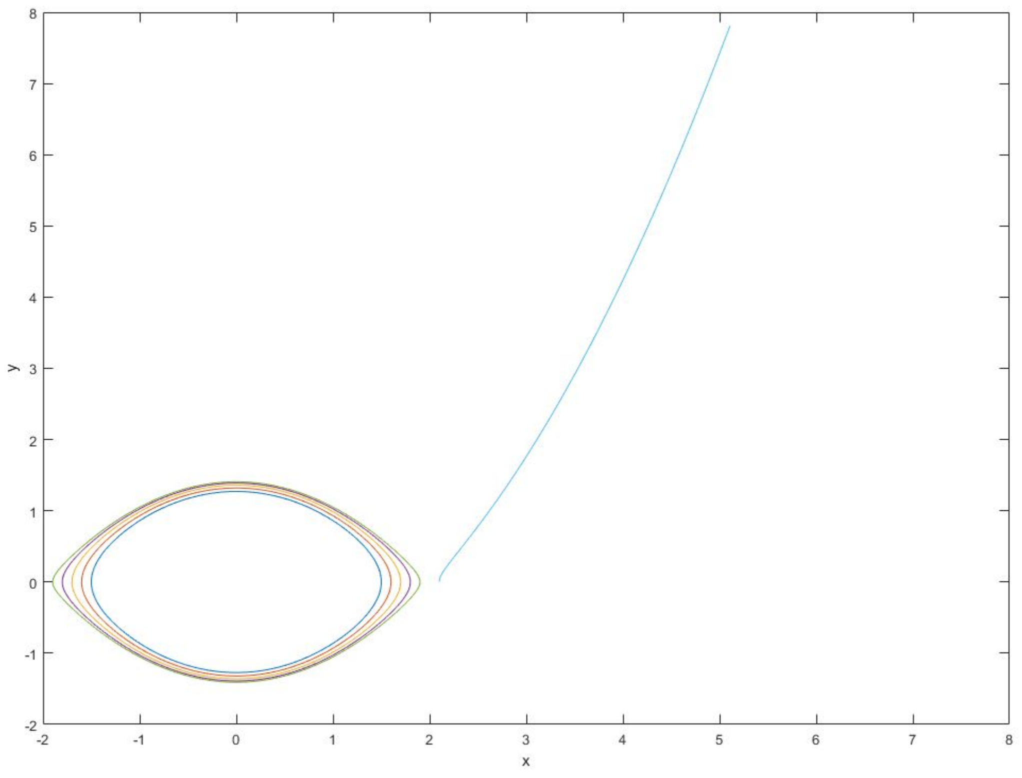

Figure 9b: *Phase plot of x(t) and y(t) for β = -0.25. Green is A=1.9, purple is A=1.8, orange is A=1.7, red is A=1.6, dark blue is A=1.5, and light blue is A=2.1.*

As can be seen in **Figure 9b**, the solution for A=2.1 is not an oval like the smaller values of A. Instead, the spring force in A=2.1 is reinforcing, and will eventually break the oscillator as time increases.

CONCLUSION

In this lab, we learned about the basic behavior of undamped, unforced linear and nonlinear oscillators. We derived mathematical models to describe the behavior of a linear in Task 2, and then the nonlinear Duffing oscillator in Tasks 1 and 3. We also analyzed how initial conditions and parameters affect the behavior of each of the two systems, enhancing our knowledge of oscillators as well as nonlinear differential equations. We then further built upon our engineering intuition and communication skills by explaining these mathematical equations in a non-technical manner. Specifically, we looked at how the period of oscillation is affected by the initial displacement for both the hardening case and the softening case of the Duffing oscillator.

We subsequently reinforced this mathematical intuition with a rigorous computational analysis based on our previously-designed adapt.m program. We modified and implemented this program to compute the average period of our oscillator using an adaptive improved euler method. In doing so, we also learned how to integrate the technique of linear interpolation into the adaptive euler method to get additional information between timesteps and find an accurate approximation to the period of oscillation. We implemented this technique within our adapt.m script in the form of several if statements. This allowed us to pinpoint times when one of the components crossed zero. Furthermore, we redesigned our prime.m script to compute the output of dx and dy at given time points, and added an additional counter to keep track of how many times this script was called. This process gave us a good chance to learn how to actively modify and apply the adaptive improved euler method to solve physical problems.

Summatively, this computational analysis yielded several critical outputs that we tabulated in Tables 1 and 2. We used numerical intuition to break down these results, and compare them to the expected mathematical results found previously. We further built upon this in a series of graphs and figures that visually analyzed this data in Tasks 4 and 5, and related those figures back to our previous mathematical and tabulated results as well. Furthermore, we related all of our results, both mathematical and computational, back to our physical intuition of an oscillator. Thus, we synthesized our mathematical, computational, and physical intuitions of an oscillator together. Ultimately, we have developed a more complete understanding and appreciation for the intricacies of oscillator modeling and computing. Finally, this lab afforded us the opportunity to work in a group. With this opportunity came the challenge of effective communication, equitable delegation of tasks, and time management.

Appendix 1

Prime.m

```
function [dx,dy]= Prime(x0,y0)

global beta

global count

dy= -(x0)-beta*(x0)^3;

dx=y0;


count=count+1;

end
```

adapt.m

```
%Set initial conditions

global count

global beta

global h

count=0;

count2 = 0;

beta= 0;

global A

A = 1;

beta=0;

x0 = A;

h=1;
```

```
hbig = 1;

hsmall = 1e-12;

hcut = 1.5;

hraise = 1.05;

tfin = 30;

epsilon=.00001;

t=0;

tmat = [];

ii = 0;

kk = 0;

tperiod = 0;

per = [];

y0=0;


%Improved Euler loop

while t<tfin

        %Go through IE Method while saving t value

        if t+h>tfin

        h= tfin-t;

        end


        %Calculate derivatives using Prime.m

        [dx,dy]= Prime(x0,y0);

        y1= y0 + h*dy;
```

```
x1= x0+ h*dx;

%Run corrector using newly calculated y and x
[dx1,dy1] = Prime(x1,y1);
yh= y0+ (h/2)*(dy+dy1);
xh= x0+ (h/2)*(y0+y1);

%If difference between y1 and tempy divided by y1 is larger than
%tolerance, h is adjusted until value is smaller than tolerance
%calculates error
k=abs(yh-y1);
k2 = abs(xh-x1);
if k2>=epsilon | k>=epsilon %if error is larger than our tolerance

h = h/hcut;
if h<hsmall %checks if new h is too small
disp 'Step size is too small'%Give error message if h is too small
%Set new values of t and yinit for next loop if there is no error
h=hsmall*1.05;

end

else
```

```
%Set new values of t

t0=t;

t=t+h;

count2=count2+1;


%Update counter used to find tzero

kk = kk+1;


%Store values needed later in a matrix

tmat(count2,1)=t;

tmat(count2,2)= xh;

tmat(count2,3)=yh;


%If y goes from positive to negative, this finds t where y was at

%zero


if kk>2   ;

if y0>0 && yh< 0 ;

tempy= y0;

tempt= t0;

        slopey = (y0-yh)/(t0-t);

        by = (tempy-slopey)*tempt;

        tzero = -by/slopey;
```

```
                    if ii>0
                    per(ii,1) = tzero-tperiod;
                    ii= ii+1;


                    else
                    ii = ii+1;
                    end
                    tperiod = tzero;
            end
            end
            %Update values for next loop
            h= h*hraise;
            x0=xh;
            y0=yh;
  end
End
avgper = mean(per);
nsteps = length(tmat)-1;
raccept = (2*nsteps)/count;


% save('c:\Desktop\x.dat','xdata','-ascii');
% save('c:\Desktop\y.dat','ydata','-ascii');
% save('c:\Desktop\xy.dat','xydata','-ascii');
% save('c:\Desktop\per.dat','perdata','-ascii');
```

```
disp(avgper);

disp(count);

disp(nsteps);

disp(raccept);


exactper = 2*pi;

errper = abs(exactper-avgper);


disp(errper);
```

Appendix 2

adaptperiod.m

```
%Set initial conditions

global count

global beta

global A

periods = [];

jj = 1;

T2 = [];

T3= [];

beta=-0.25;

hbig = 1;

hsmall = 1e-12;

hcut = 1.5;

tfin = 2.5;

epsilon=.00001;

tmat = [];

per = [];


%Start for loop that will run the code for each value of A

for A = .1:.1:1.9

%Set initial values within the loop that need to be reset upon each for loop completed

h=1;

hraise = 1.05;
```

```
x0 = A;

count=0;

count2 = 0;

ii = 0;

kk = 0;

tperiod = 0;

y0=0;

t=0;


%Improved Euler loop

while t<tfin

        %Go through IE Method while saving t value

        if t+h>tfin

        h= tfin-t;

        end


        %Calculate derivatives using Prime.m

        [dx,dy]= Prime(x0,y0);

        y1= y0 + h*dy;

        x1= x0+ h*dx;


        %Run corrector using newly calculated y and x

        [dx1,dy1] = Prime(x1,y1);

        yh= y0+ (h/2)*(dy+dy1);
```

```
xh= x0+ (h/2)*(y0+y1);


%If difference between y1 and tempy divided by y1 is larger than

%tolerance, h is adjusted until value is smaller than tolerance

%calculates error

k=abs(yh-y1);

k2 = abs(xh-x1);

if k2>=epsilon | k>=epsilon %if error is larger than our tolerance


h = h/hcut;

if h<hsmall %checks if new h is too small

disp 'Step size is too small'%Give error message if h is too small

%Set new values of t and yinit for next loop if there is no error

h=hsmall*1.05;


end


else


%Set new values of t

t0=t;

t=t+h;

count2=count2+1;
```

```
%Update counter used to find tzero

kk = kk+1;


%Store values needed later in a matrix

tmat(count2,1)=t;

tmat(count2,2)= xh;

tmat(count2,3)=yh;


%If y goes from positive to negative, this finds t where y was at

%zero


if kk>2   ;

if y0>0 && yh< 0 ;

tempy= y0;

tempt= t0;

        slopey = (y0-yh)/(t0-t);

        by = (tempy-slopey)*tempt;

        tzero = -by/slopey;


        if ii>0

        per(ii,1) = tzero-tperiod;

        ii= ii+1;


        else
```

```
                ii = ii+1;
                end
                tperiod = tzero;
        end
        end
        %Update values for next loop
        h= h*hraise;
        x0=xh;
        y0=yh;
  end
end

%Store periods in matrix
periods(jj,1)=A;
periods(jj,2)=avgper;
save('c:\Desktop\periods.dat','periods','-ascii')

%Calculate approximations and store them
gamma = beta * A^2;
T2calc = (2*pi)*(1-((3/8)*gamma));
T3calc = (2*pi)*(1-((3/8)*gamma) + (57/265)*gamma^2);

T2(jj,1) = T2calc;
T3(jj,1) = T3calc;
```

```
%Update counter used for storing approximations

jj = jj+1;

end
```