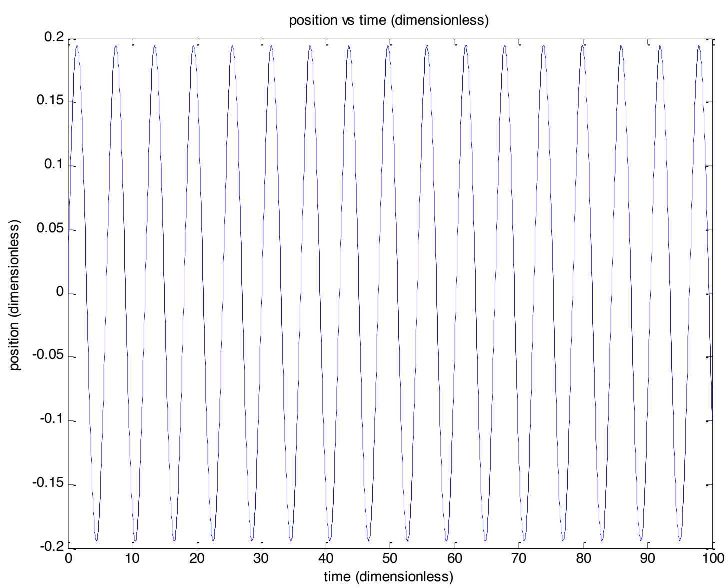
V. Chiu

Below: all time steps are of 5.00000E-002.

Below:

Using gamma = 3.0;

initial_v = 0.2;

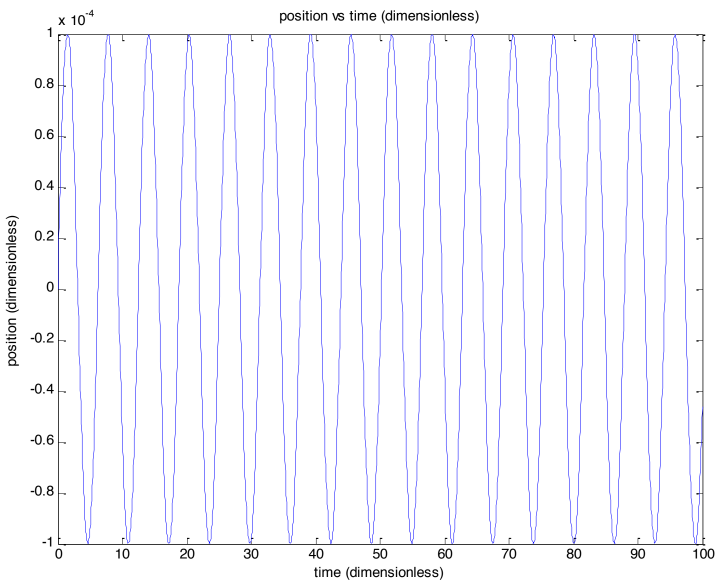


Below:

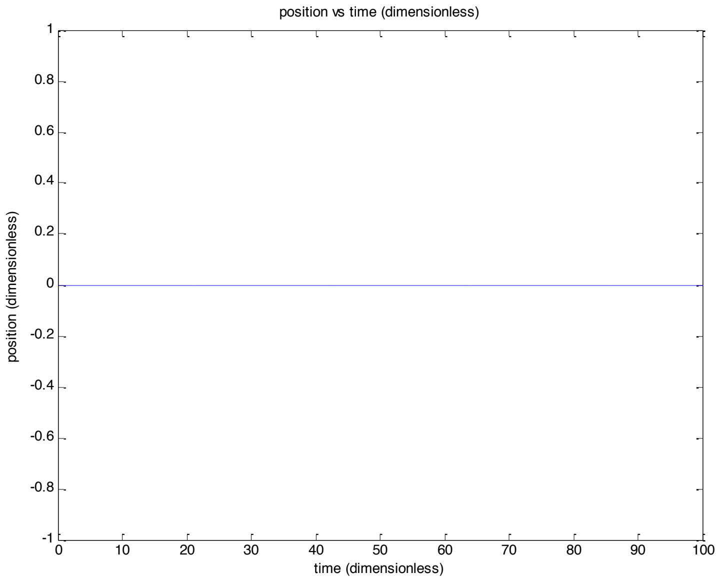
Gamma = 3.0

initial_x = 0.0;

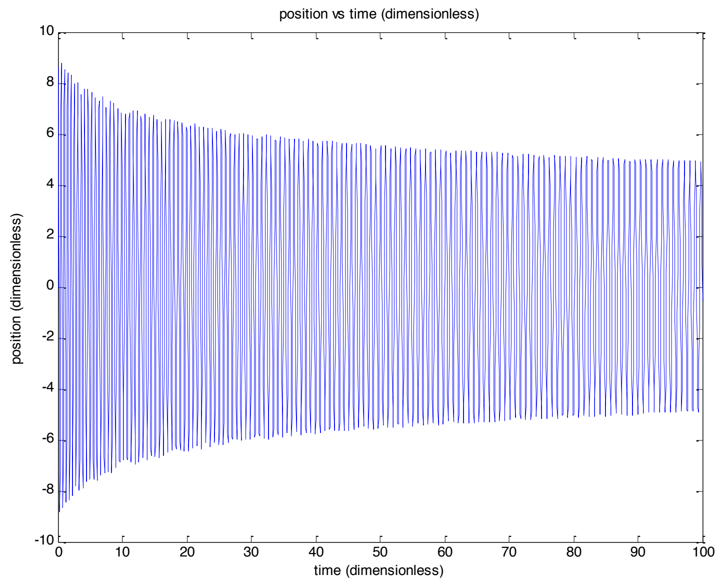
initial_v = 0.0001;



Below v = 0;


position vs time (dimensionless)
position (dimensionless)
time (dimensionless)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50
60
70
80
90
100

V= 100



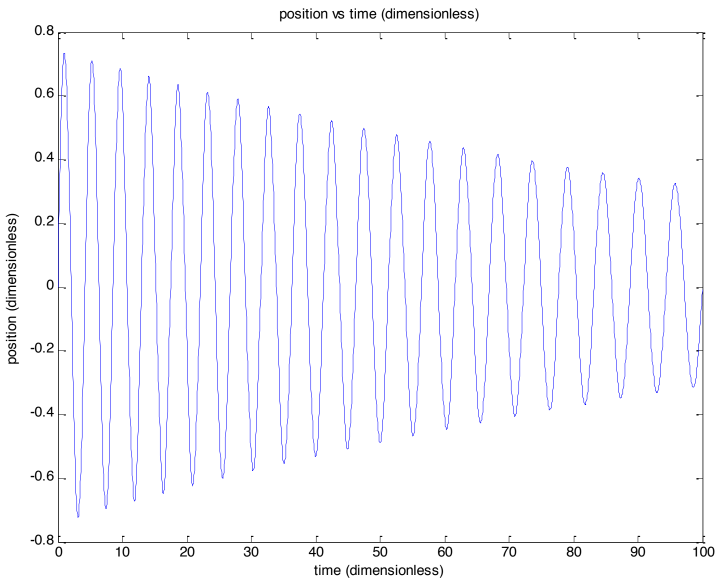
Frequency increases proportionally to the increase in initial velocity with initial position fixed at 0 and constant gamma.

Q1b)

```
double period_initial = 10.0;

final_t = period_initial*10.0;

h = 0.05;

number_of_steps = final_t/h;

initial_x = 0.0;

initial_v = 1.0000;

double damp =0.02;
```

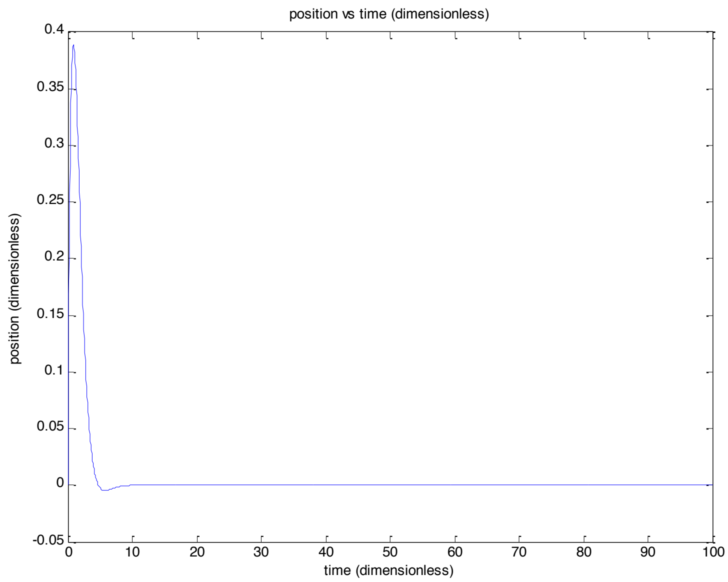
```
dydt[0] = y[1];

dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1];
```

below is underdamped



Below is underdamped: double damp =1.7;

```
dydt[0] = y[1];

dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1];
```



Below is overdamped:

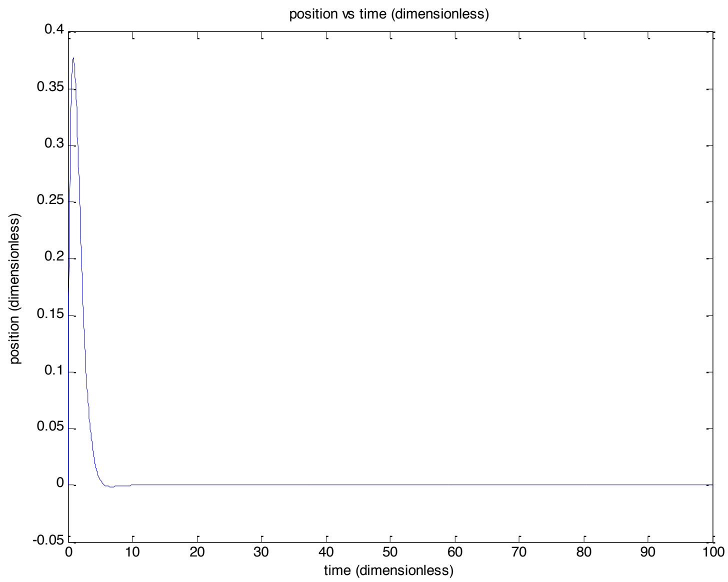
initial_x = 0.0;

initial_v = 1.0000;

double damp =1.8;

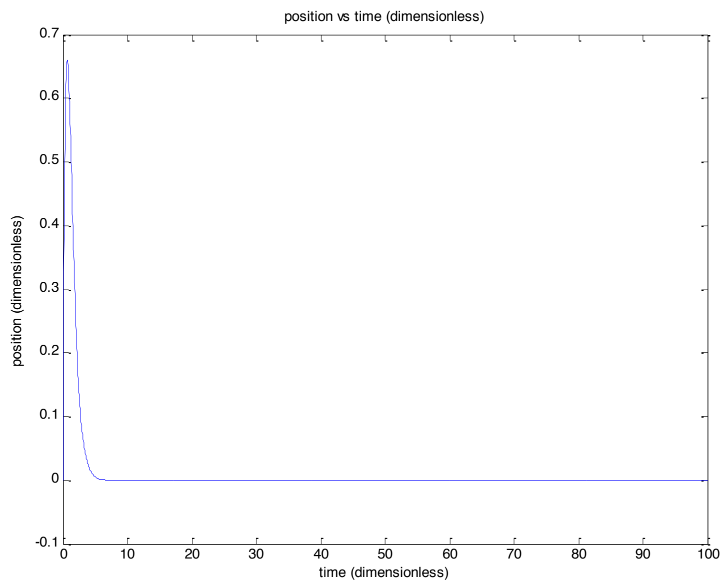
dydt[0] = y[1];

dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1];



Below; initial v is increased from 1 to 2:

```
double damp =2.0;

   dydt[0] = y[1];

   dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1];
```

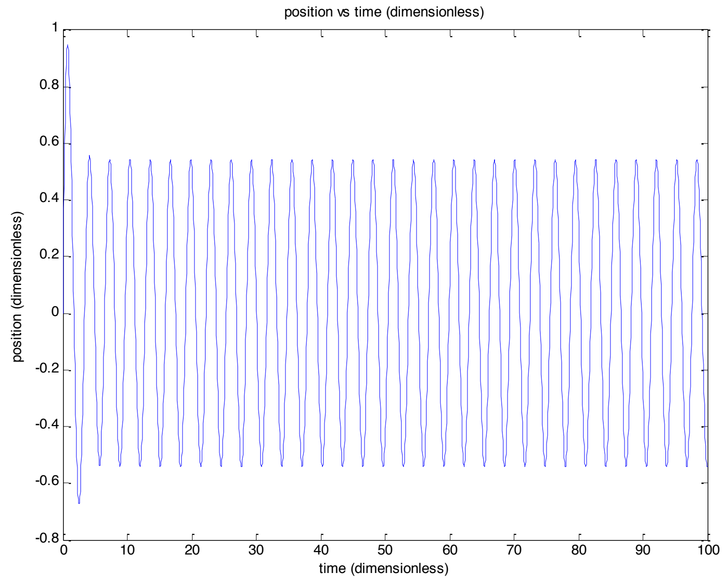


The value that results in critical damping increases from 1.8 to 2.0 when changing initial v from 1.0 to 2.0.

Q1c)

Below:

```
double damp =2.0;


   double f = 2.5;

   double omega = 2.0;

   dydt[0] = y[1];

   dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1]+f*cos(omega*t);
```



Program code for part c below:

```
//Code modified by Vincent Chiu.  Based on code by professor Kristin Schleich.

//Pendulum with NO Driving force, damping of 10.0.

//duffing oscillator final project


#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <assert.h>
```

```
void initialise (double *, double *, double *, int *);

void derivatives(double, double *, double *);

void euler(double *, double *, int, double, double, double *);

void output( FILE *, double, double, double *, double);

void runge_kutta_2(double *, double *, int, double, double, double *, void (*)(double, double*,
double*));

void runge_kutta_4(double *, double *, int, double , double , double *, void (*)(double, double *, double
*));


int main()

{

   printf("Welcome to Chiu Industries.  Pendulum Calculator.  NO Driving Force Mode. \n");

   FILE *output_file;

   //declarations of variables

   int number_of_steps;

   double initial_x, initial_v, final_t, E_initial;

   double h;

   double y[2], dydt[2], yout[2];

   double t;


   output_file = fopen("finalq1c.dat", "w");


   //read in input data from screen


   //initialise (&initial_x, &initial_v, &final_t, &number_of_steps);
```

```
// double length_rod = 1.0;

// double gravitational_constant = 1.0;


// double angular_frequency_initial = sqrt(gravitational_constant/length_rod);

//double period_initial = 2.0*M_PI/angular_frequency_initial;



double period_initial = 10.0;

final_t = period_initial*10.0;

h = 0.05;

number_of_steps = final_t/h;

initial_x = 0.0;

initial_v = 2.0000;


//number_of_steps = 100000;




assert(number_of_steps >0);


//initialise position and velocity


y[0] = initial_x;

y[1] = initial_v;
```

```
    E_initial = 0.5*y[0]*y[0] + 0.5*y[1]*y[1];



    //h=final_t/number_of_steps;

    output(output_file, h, t, y, E_initial);



    t=0.0;

    while(t<=final_t)

    {

        derivatives(t,y,dydt);

        //euler(y, dydt, 2, t, h, yout);

        runge_kutta_4(y, dydt, 2, t, h, yout, derivatives);

        y[0]=yout[0];

        y[1]=yout[1];

        output(output_file, h, t, y, E_initial);

        t+=h;

    }

    fclose(output_file);

    return 0;



} //end main program


void initialise(double *initial_x, double *initial_v, double *final_t, int *number_of_steps)
```

```
{

    printf("Read in from screen the initial position x, initial velocity v, final time and number of steps \n");
    scanf("%lf %lf %lf %d", initial_x, initial_v, final_t, number_of_steps);
    return;

} // end of function initialise



// this function provides the first derivative of y[0] and y[1]; i.e.
// it provides the rhs of the couple first order equations of motion
//for the harmonic oscillator with omega=1

void derivatives(double t, double *y, double *dydt)
{

    //double damping_constant = 2.0;
    //double length_rod = 1.0;
    //double gravitational_constant = 1.0;
    //double mass = 1.0;
    // double angular_frequency_initial = sqrt(gravitational_constant/length_rod);
    // double b = damping_constant*angular_frequency_initial/(mass*gravitational_constant);



    //dydt[0] = y[1];
```

```
   //dydt[1] = -y[0];
//dydt[1] = -b*y[1]-sin(y[0]);
   //dydt[1] = -y[0]-3.0*pow(y[0],3);



   double damp =2.0;

   double f = 2.5;
   double omega = 2.0;
   dydt[0] = y[1];
   dydt[1] = -y[0]-3.0*pow(y[0],3)-damp*y[1]+f*cos(omega*t);

}// end of function derivatives

// This function computes the first derivative with centered algorithm

void euler(double *y, double *dydt, int n, double t, double h, double *yout)
{
   int i;
   for(i=0; i<n; i++)
   {
      yout[i]=y[i] + h*dydt[i];
   }
}  // end of euler integrator
```

```
}

// function to write out the final results

void output(FILE *output_file, double h, double t, double *y, double E_initial)
{
   fprintf(output_file, "%12.5E \t %12.10E \t %12.10E \t %12.10E \t %12.10E \n", h, t, y[0], y[1],
0.5*y[0]*y[0]+0.5*y[1]*y[1]-E_initial);
   return;

}//end of function output


void runge_kutta_2(double *y, double *dydt, int n, double x, double h, double *yout, void
(*derivatives)(double, double*, double *))
{

   int i;
   double xh;
   double *dyt, *yt;
   // allocate space for local vectors
   dyt = (double *)malloc(n*sizeof(double));
   yt = (double *)malloc(n*sizeof(double));
   xh = x+h/2.0;
   for (i=0; i<n; i++)
   {
      yt[i] = y[i]+h/2.0*dydt[i];
   }
```

```
//computation of y_t+1/2h = yt +h/2 dy/dt using k1 = h dy/dt

   (*derivatives)(xh,yt,dyt);

// find k2 which is h* dyt


   for(i=0; i<n; i++)

   {

      yout[i] = y[i] +h*dyt[i];

   }// increment yout using yout = y+h k2

   free(dyt);

   free(yt);


}//end of function runge katta 2






void runge_kutta_4(double *y, double *dydx, int n, double x, double h, double *yout, void
(*derivatives)(double, double *, double *))

{

   int i;

   double     xh,hh,h6;

   double *dym, *dyt, *yt;

   //   allocate space for local vectors

   dym = (double *) malloc(n*sizeof(double));

   dyt =  (double *) malloc(n*sizeof(double));
```

```
yt =  (double *) malloc(n*sizeof(double));
hh = h*0.5;
h6 = h/6.;
xh = x+hh;
for (i = 0; i < n; i++)
{
   yt[i] = y[i]+hh*dydx[i];
}
(*derivatives)(xh,yt,dyt);     // computation of k2, eq. 3.60
for (i = 0; i < n; i++)
{
   yt[i] = y[i]+hh*dyt[i];
}
(*derivatives)(xh,yt,dym); //  computation of k3, eq. 3.61
for (i=0; i < n; i++)
{
   yt[i] = y[i]+h*dym[i];
   dym[i] += dyt[i];
}
(*derivatives)(x+h,yt,dyt);    // computation of k4, eq. 3.62
//      now we upgrade y in the array yout
for (i = 0; i < n; i++)
{
   yout[i] = y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);
}
```

```
   free(dym);

   free(dyt);

   free(yt);

}      //  end of function Runge-kutta 4
```