Approximations in physics

Ved Patel 67 Vijay Panchal 68

December 2, 2022

Contents

1 Introduction

Approximation method is yet most essential topic in physics. Also, approximation is yet essential trick for physicists. Physicists love to do approximations, like in functional expansion for getting polynomials for their ease or may be specialized idealization in particular topic. Approximation help them to **seeing through physics** instead of going in to maze of exactness in formidable mathematics. Getting interpretation or i say knowing system is sometime more important then going for regorious mathematics. For example, famous equation of fluid dynamics **Navier-Stoke equation** can be imposible to solve but as physicist they know what it is.

Be aware, that approximation is just approximation. We should remember everytime we do that. Sometime we forgot actual system which is far from ideal. We should know that we are on mission to know nature not just building new theories.

Let's dive into one example, that showes implication of approximations. In classical mechanics, we have some major theories, one is of oscillatory motions. In Oscillation theory we studied **Simple Harmonic Oscillation**, but as we are going to see that simple harmonic oscillation is not exactly that simple without approximation. We had actually changed whole system unknowingly, but beauty of physics is that it is still help to understand concept and motion of it.

Before that we would like to show gratitude to literature that helped us to finish this project. Firstly, for our classical mechanics textbooks classical mechanics by goldstein [2] and by J.C. Upadhyay [10]. Then book on differential equations [1] and [5]. Estimation of parameters of damping forces [8].

2 Example of Approximation

Approximation is used in almost every branch of physics, not just physics but every field of sciences. We are going to give profound example of understanding advantages and disadvantages of approximation.

2.1 Defining our problem

We learned simple pendulum from very starting of physics course. But what if i say that simple pendulum is not actually simple in sense that approximation hide most of things away from our eyes to see.

Let's take one pendulum,by taking length $l = 1m$ string (assuming non deforming) attaching to bob of mass $m = 0.1kg$. String attached to rigid wall as shown in figure. Then we give it a initial deviation as θ initial θ_0. Initial velocity of system $v_0 = 0$. We have accelaration of $g = 9.8$ downward.

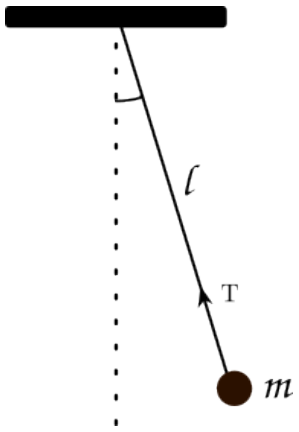

Figure 1: pendulum with string lenth l and mass m

For understanding consequences of approximation, we took simulations by solving both equation of motion (approximated and exact). For getting equations of motion we used **Newtonian formulation** which is quite easy to work with in this type of problems, since we are working with **non conservative** system.

2.2 Pendulum motion in presence of damping force

In real situations we have non-conservative forces affecting on system. In our system we have air resistance acting on bob. This drag force will always be proportional to it's physical shape and size. Since, we can't tell exactly drag force mathematically, We have **two often used approximations**. Firstly, there is stockes's law of drag which is linearly proportional to the velocity. Then we have Newton's drag law which is quadratically proportional to the velocity.

We will see that how this two approximation affect our system and it's simulation. We also see mathematical forms of this two but firstly, let's define this two damping coefficient.

2.2.1 Damping by linear to the velocity approximation

Let's discuss coffiecient of linear damping. Linear to the velocity drag is also called stoke's drag. We can write equation of damping by $F_{damping} = k_l v$. Where, k_l is damping coefficient, value of it depend on shape of object. For our spherical bob it has for of $k_l = 3\pi\eta d$, where η is dynamic viscosity of medium and have values of $\eta = \rho\nu$. We have d as diameter of sphere, for our purpose we are using as $d = 0.01m$.

2.2.2 Damping by quadratic to the velocity approximation

We used quadratic damping approximation as nonlinear damping approximation. This is also called newton's drag and it's better approximation then linear one. As quantitatively it can work with turbulent flow with 1000 to 100000 reynolds number. Where linear approximation works reynold's number upto 1. Quadratic damping is better than linear approximation but we can't have close form solution, it is the reason we use linear damping approximation. In quadratic damping it has following form, $F_{damping} = k_q v^2$. Here, $k_q = \frac{1}{8} c_d \rho \pi d^2$, where c_d is drag coefficient and has value of 0.47. ρ and d are density of fluid and diameter.[6][3]

2.3 General equation motion

Let's derive equation of motion for our system which can be modified as we took approximations in it. We have Figure 1 showing pendulum with unextensible string of length l with sphere of diameter d and mass m. Our system is in fluid with density ρ and viscosity η. Let's use newton's law of motion to derive equation of motion,

First of all, we compared horizontal and vertical forces.

$$F_{damping} cos(\theta) - T sin(\theta) = ma_x \tag{1}$$

$$F_{damping} sin(\theta) + T cos(\theta) - mg = ma_y \tag{2}$$

Adding equation 1 and equation 2 with multiplication by $cos(\theta)$ and $sin(\theta)$ respectively.

$$F_{damping} sin^2(\theta) + F_{damping} cos^2(\theta) - mgsin(\theta) = ma_x cos(\theta) + ma_y sin(\theta)$$

$$F_{damping} - mgsin(\theta) = m(asin^2(\theta) + acos^2(\theta))$$

$$F_{damping} - mgsin(\theta) = ma \tag{3}$$

From,

$$a = (\ddot{r} - r\dot{\theta}^2)\hat{r} + (r\ddot{\theta} + 2\dot{r}\dot{\theta})\hat{\theta}$$

Where, $r = l$ and since $\dot{l} = 0$, $a = l\ddot{\theta}$. So, equation 3 becomes,

$$F_{damping} - mgsin(\theta) = ml\ddot{\theta} \tag{4}$$

This is **exact equation of motion**. Which will be **second order non linear equation**. Finding it's exact solution is another ordeal. Let's take our approximations and cases for it.

2.4 Approximation of equation of motion : Linear differential equation with linear damping

In class, we approximated equation 4 as $\theta \to 0$ as $sin(\theta) \to \theta$. Consequently, this equation becomes very easy to solve. Also, damping force will be,

$$F_{damping} = -k_l v$$

$$F_{damping} = -k_l l\dot{\theta}$$

So, equation 4 becomes,

$$\ddot{\theta} + \frac{k_l}{m}\dot{\theta} + \frac{g}{l}\theta = 0 \tag{5}$$

$$\ddot{\theta} + \Gamma\dot{\theta} + w_0^2\theta = 0 \tag{6}$$

Where, we took $\Gamma = \frac{k_l l}{m}$ and w_0^2.

We can solve this linear equation 6 by usual methods of linear differential equation. Simply taking $\theta = e^{\lambda t}$, which gives polynomials of second order.

$$\lambda^2 + \Gamma\lambda + w_0^2 = 0 \tag{7}$$

We can find roots of this quadratic equation.

$$\lambda = \frac{-\Gamma}{2} \pm \frac{\sqrt{\Gamma^2 - 4w_0^2}}{2} \tag{8}$$

$$\lambda = \frac{-\Gamma}{2} \pm \sqrt{\frac{\Gamma}{2}^2 - w_0^2} \tag{9}$$

Here we getting three type of roots,

1. Roots where $\frac{\Gamma}{2} = w$. this is **critical damping condition**, where we getting $\lambda = \frac{-\Gamma}{2}$. Putting λ into our solutions, $\theta = e^{\frac{-\Gamma}{2}t}$. Which suggest this will only decay with time and never overshoots from equilibrium position. Which is desired in certain condition but not for us.

2. Roots where $\frac{\Gamma}{2} > w$. this is **overdamping condition**, where we getting $\lambda = \frac{-\Gamma}{2} \pm \sqrt{\frac{\Gamma}{2}^2 - w_0^2}$. So from here we get $\theta = e^{\frac{-\Gamma}{2}t} e^{\pm\sqrt{\frac{\Gamma}{2}^2 - w_0^2}t}$. This also have exponential term in it which will only decay with time and never overshoots from equilibrium position.

3. Roots where $\frac{\Gamma}{2} < w$. this is **underdamping condition**, here $\lambda = \frac{-\Gamma}{2} \pm i\sqrt{w_0^2 - \frac{\Gamma}{2}^2}$. $\theta = e^{\frac{-\Gamma}{2}t} e^{\pm i\sqrt{w_0^2 - \frac{\Gamma}{2}^2}t}$. This has complex term, which implicitly suggest that it'll overshoot and oscillate. This our topic of interest for this project.

Without forgetting our initial system we came to we took third case as our solution.

$$\theta = e^{\frac{-\Gamma}{2}t} e^{\pm i\sqrt{w_0^2 - \frac{\Gamma}{2}^2}t}$$

Taking $w^2 = w_0^2 - \frac{\Gamma}{2}^2$. And writing our solution in linear combination from above equation,

$$\theta = e^{\frac{-\Gamma}{2}t}(C_1 e^{iwt} + C_2 e^{-iwt}) \tag{10}$$

Taking real part of equation 10. Since it'll represent real motion of system. At last we get equation like this,

$$\theta = e^{\frac{-\Gamma}{2}t} A cos(wt - \delta) \tag{11}$$

Where, A and δ can be find from initial conditions and $w = \sqrt{w_0^2 - \frac{\Gamma}{2}^2}$.

2.5 Non linear equation of motion with linear damping

In equation 4 we can write linear damping term without taking approximation as $sin(\theta) \to \theta$,

Writing again 4,

$$F_{damping} - mgsin(\theta) = ml\ddot{\theta}$$

Here, putting $F_{damping} = -k_l l\dot{\theta}$ will give us,

$$\ddot{\theta} + \frac{k_l}{m}\dot{\theta} + \frac{g}{l}sin(\theta) = 0 \tag{12}$$

This is second order nonlinear equation we can't get it's closed form solution but we can get numerical one. Let's make it easy to use in numerical methods.

Take $\phi = \dot{\theta}$ and $\frac{k_l}{m} = \Gamma$. So, equation 12 becomes,

$$\dot{\phi} + \Gamma\phi = -\frac{g}{l}sin(\theta) \tag{13}$$

We can use numerical methods like Runge-Kutta method to solve this equation. I have given brief overview of runge kutta methods in appendix 1. For that we define ϕ and $\dot{\phi}$ as following,

$$\phi = \dot{\theta} \tag{14}$$

$$\dot{\phi} = -\Gamma\phi - \frac{g}{l}sin(\theta) \tag{15}$$

We have two complementary equations 14 and 15. We can use now Runge-Kutta method on it. I have used in my code file *main.py*, you can access it in appendix.

2.6 Simulations of the two equations

I have done nice simulation which give hands on experience of two equation, both have very similar results when θ is very small, again understandable as $\theta \to 0$ we can approximate $sin(\theta) \to \theta$. But when θ increase slightly we have massive changes in solution with time. Let's look at $\theta = \frac{\pi}{10}$, (here, we take viscosity of air at $(183438 \pm 0.35) \times 10^7$ c.g.s. units. [7])

Initially both are same as you can see in pictures (at $t = 0$),

Now, as we look with increment in time we can see it deflecting slightly with it. This is picture at $t = 100s$,

Here, red bob is exact solution (nonlinear equation) where green is linear equations solution.

For more information, i gave all my code in appendix and also in my github page vijaypanchalr3.

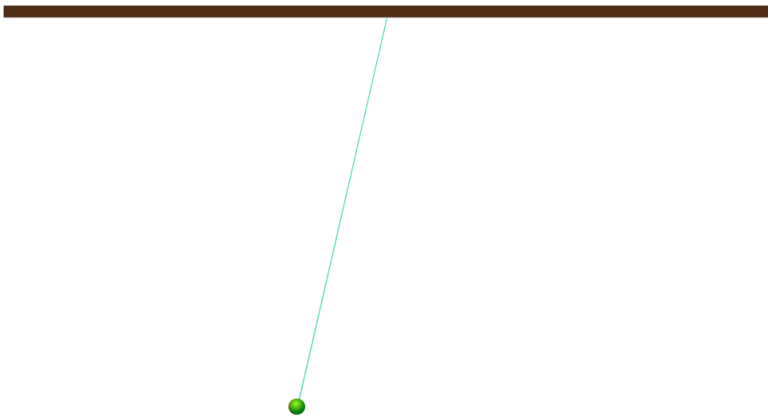

Figure 2: pendulum at $t = 0s$

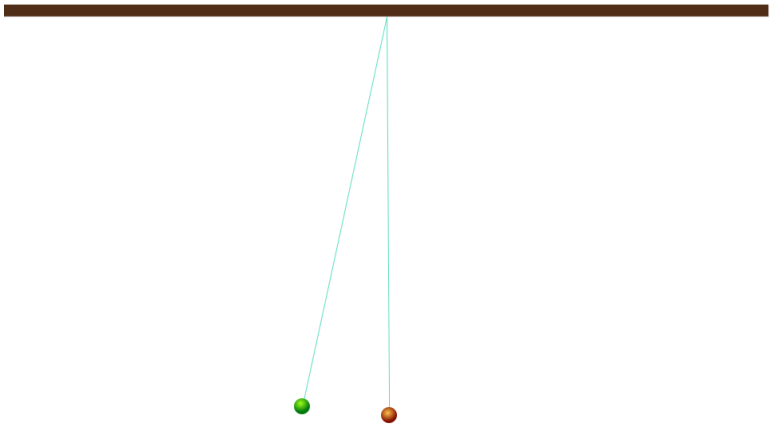

Figure 3: pendulum at $t = 100s$

2.7 Equation of motion with quadratic damping

Quadratic damping gives better approximations over as linear damping but also gives burden of nonlinearization. In equation 4 we can put value of quadratic damping approximation.

$$F_{damping} - mgsin(\theta) = ml\ddot{\theta}$$

We can put value of damping force as $F_{damping} = -k_q l^2 \theta^2$,

$$-k_q l^2 \theta^2 - mgsin(\theta) = ml\ddot{\theta}$$

$$\ddot{\theta} + \frac{k_q l}{m}\theta^2 + \frac{g}{l}sin(\theta) = 0$$

$$\ddot{\theta} + \Gamma_q \theta^2 + \frac{g}{l}sin(\theta) = 0$$

This is also second order differential. Even with approximation this have still no close form solution. We will solve this numerically in simulation just see difference between linear damping.

3 Results and conclusion

As we seen earlier our simulations discuss how exact solution differs from approximation. Also, we see that linear damping is still a approximation and can be replaced by better approximation when medium have turbulant flow by bob of our pendulum.

3.1 Effect on angular frequency of pendulum with approximation and exact solution

As we have seen in our simulation that in linear solution equation 11, we got single constant frequency with θ but not in nonlinear one. Nonlinear equation has non constant frequency. We can see from graph of it that how it differs from constant frequency of linear solution. [9][4]

3.2 For different θ_0 it has different $\theta(t)$

Let's look at how our two solution exactly changes with time. For that we took values of θ on time scale and changed it's θ_0.

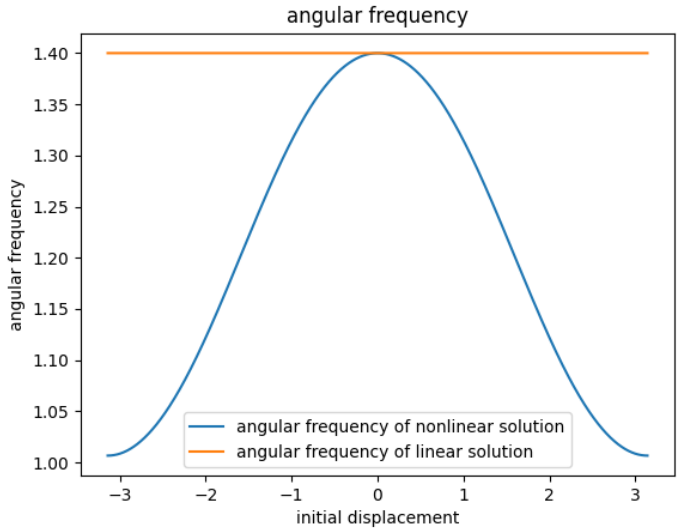

Figure 4: How angular frequency change with θ_0 in nonlinear solution and shows constant at value $\sqrt{\frac{g}{l}}$

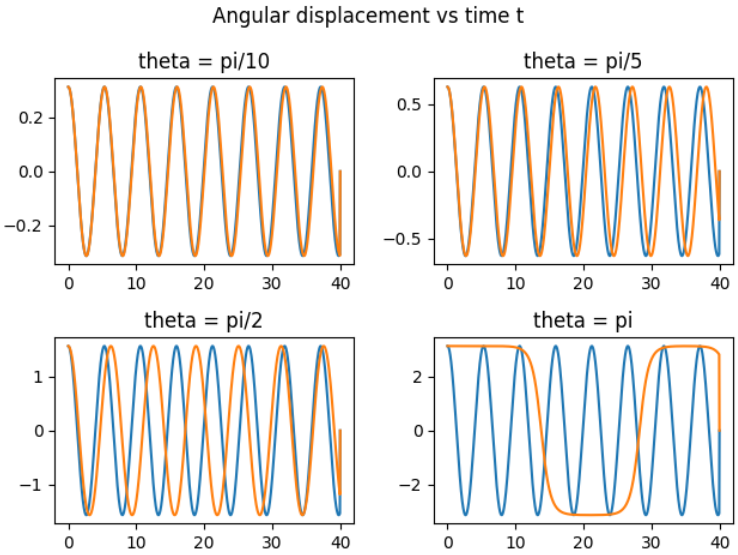

Figure 5: For different θ we have completely different type of solution

3.3 Hidden informations

Approximation not only affect in quantitatively but also sometimes qualitatively. Let's discuss part of that lost in approximation. Phase planes are geometrical procedures of obtaining properties of system (here solutions). If write our equation of motion for the pendulum from 4.

$$F_{damping} - mgsin(\theta) = ml\ddot{\theta}$$

Here, $F_{damping}$ is velocity depend term. Also, velocity $v = l\dot{\theta}$. So, we can write equation 4 in following way,

$$F_d(\dot{\theta}) - mgsin(\theta) = ml\ddot{\theta}$$

Taking $\phi = \dot{\theta}$ led to,

$$\dot{\phi} = (\frac{1}{ml})F_d(\phi) - \frac{g}{l}sin(\theta) \tag{16}$$

$$\dot{\theta} = P(\theta, \phi) \tag{17}$$

$$\dot{\phi} = Q(\theta, \phi) \tag{18}$$

Dividinge equation 18 and 17 will give us,

$$\frac{\dot{\phi}}{\dot{\theta}} = \frac{\frac{d(\phi)}{dt}}{\frac{d\theta}{dt}} = \frac{d\phi}{d\theta} = \frac{Q(\theta, \phi)}{P(\theta, \phi)} \tag{19}$$

This equation 19 will give **phase trajectory** of our system. By, equation 16 value of ϕ gives,

$$\frac{d\phi}{d\theta} = \frac{(\frac{1}{ml})F_d(\phi) - \frac{g}{l}sin(\theta)}{phi} \tag{20}$$

In equation 20 if we put value of $F_d = k_l l\dot{\theta} = k_l l\phi$ for linear damping damping force, we will get,

$$\frac{d\phi}{d\theta} = \frac{\Gamma\phi - \frac{g}{l}sin(\theta)}{phi} \tag{21}$$

This is our equation of motion which give trajectory for exact solution and similarly approximated solution would be,

$$\frac{d\phi}{d\theta} = \frac{\Gamma\phi - \frac{g}{l}\theta}{phi} \tag{22}$$

Ploting this two will give two distinct phase planes which gives important properties of system. Two phase plane plots are as following,

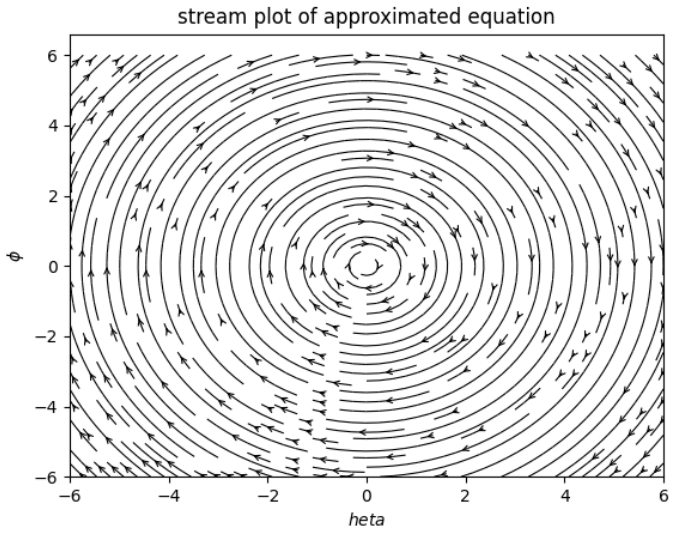

Figure 6: This is phase plane of approximate solution where we took $sin(\theta) \rightarrow \theta$. here you can see big vortex.

As we can see that in approximated equation we have only one big vortex. In contrast to that we have vortexs with lines.

In first figure in approximated equation we have phase trajectory of simplest, spiral. This suggest a simple harmonic motion which will always have stable and periodic motion.

In second figure we got completely different phase trajectory. We got three are with completely different behavior of system. The three cases of it are following,

In first area, which is vortex like previous has stable and periodic with energy $E < 2mgl$, second area we got **seddle point**, in which we got unstable and aperiodic motion ($E = 2mgl$). In last area which is upper and lower part of plot are just lines, this are shows circular motion around pivot of string with wall. Energy for this kind of motion is $E > 2mgl$.

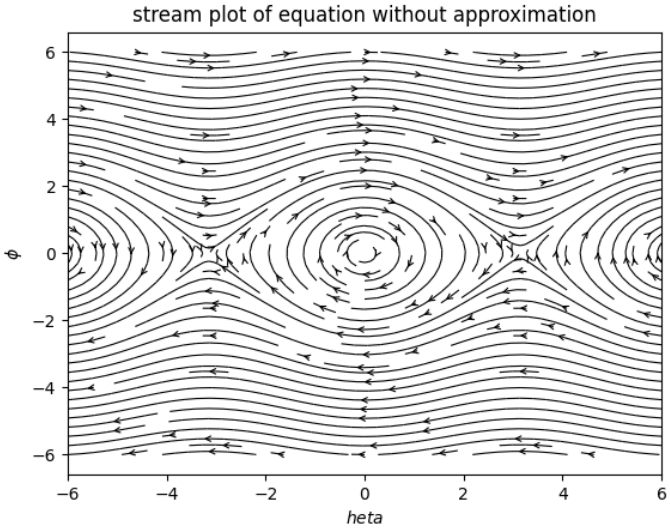

Figure 7: This is phase plane of exact solution without previous approximation.

4 Appendix

4.1 (Appendix:1) Understand Runge-Kutta method

In our this simulation we made use of Range Kutta fourth order method as numerical method for solving non-linear differential equation and linear differential equation with it. So, it is good idea to understand what is Range-Kutta fourth order method and how can we implement to solve present differential equations.

Runge Kutta Method is not predictor-corrector method like other numerical method (namely, modified Euler method, Adams-Bashmoth-Moulton method) for solving differential equation. It uses four different new variables and then simply addition and multiplication predict our initial value problem with good accuracy.

We can use RK method in following manor,

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$z_{n+1} = z_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4)$$

Where,

$$k_1 = hf(x_n, y_n, z_n)$$

$$l_1 = hg(x_n, y_n, z_n)$$
$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2})$$
$$l_2 = hg(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2})$$
$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2})$$
$$l_3 = hg(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2})$$
$$k_4 = hf(x_n + h, y_n + k_3, z_n + l_2)$$
$$l_4 = hg(x_n + h, y_n + k_3, z_n + l_2)$$

Where, h is step and f, g are two complimetry function, in our purpose we used $f = \theta$ and $f = \phi$.

4.2 (Appendix:2) Simulation code

Now, come animation part. Which we basically used **pygame** in **python**. We first get array of both solutions with interval of $\frac{1}{60} second$ and give this data in position function in my *main.py* file which just use convert each to the Cartesian coordinates from initial Polar coordinate. This is because *pygame* screen rectangular coordinates with units in pixel of screen.

Following data, we used as constant which i defined in *constant.py* file, as per close inspection you can see that we used C.G.S. units because of better visual on computer screen. Remember, we made this code for reconstruct purpose only.

My *constant.py* file

```
from math import sqrt

# defining constants in C.G.S.

pi = 3.141592
width,height = 1360,720 # pygame window size in pixel units
origin_x,origin_y = width/2,height/8 # setting up the origin O

# density
rho = 0.001293

# newton's drag coefficient
cd = 0.47

# diameter
d = 0.05
```

```
# mass
m = 500

# length of string
l = 500

# gravitation accelaration
g = 980

# viscosity of air
eta = 0.0001834

kl = 3*pi*eta*d
kq = (1/8)*pi*cd*rho*d*d

gammal = (kl*l)/m
gammaq = (kq*l)/m
w0 = sqrt(g/l) # natural frequncy of SHM
theta_initial = 3.141592/10 # initial theta in radian
radius = 10 # radius of ball in pixel
fps = 120 # frame per second
```

This is my *main.py* file, in which i defined all functions for calculations. In which, i have Runge-Kutta method defined and solution and also phase planes defined.

```
from constants import *
from numpy import sin, sqrt, zeros

def f2nonlinear_linear(theta,phi): # we defined second auxillary equation
    from nonlinear term.
    return -((gammal)*phi)-(w0*sin(theta))

def f2linear_linear(theta,phi): # we defined second auxillary equation
    from linear term.
    return -((gammaq)*phi)-(w0*theta)

def f2llinear_nonlinear(theta,phi): # we defined second auxillary equation
     from linear term.
    return -((kq)*phi*phi)-(w0*theta)

def f2nonlinear_nonlinear(theta,phi): # we defined second auxillary
    equation from linear term.
    return -((kq)*phi*phi)-(w0*sin(theta))
```

```
# range-kutta method defined
def RK4(theta,phi,h,K):
    h = h/8
    for i in range(8):
        k1 = h*phi
        l1 = h*K(theta,phi)
        k2 = h*(phi+(l1*0.5))
        l2 = h*(K(theta+(k1*0.5),phi+(l1*0.5)))
        k3 = h*(phi+(l2*0.5))
        l3 = h*(K(theta+(k2*0.5),phi+(l2*0.5)))
        k4 = h*(phi+l3)
        l4 = h*(K(theta+k3,phi+l3))
        k_ = (1/6)*(k1+k4+2*(k2+k3))
        l_ = (1/6)*(l1+l4+2*(l2+l3))
        theta+=k_
        phi+=l_
    return theta,phi

# Solutions of linear term ---- gives array of length (Total_time*fps)
def linear_linear(theta_initial,Total_time,fps):
    linear_solutions = zeros([Total_time*fps+2])
    linear_solutions[0] = theta_initial
    phi = zeros([Total_time*fps+2])
    phi[0],t = 0,0
    while t<Total_time*fps:
        linear_solutions[t+1], phi[t+1] = RK4(linear_solutions[t],phi[t],1/
            fps,f2linear_linear)
        t+=1
    return linear_solutions

def linear_nonlinear(theta_initial,Total_time,fps):
    linear_solutions = zeros([Total_time*fps+2])
    linear_solutions[0] = theta_initial
    phi = zeros([Total_time*fps+2])
    phi[0],t = 0,0
    while t<Total_time*fps:
        linear_solutions[t+1], phi[t+1] = RK4(linear_solutions[t],phi[t],1/
            fps,f2llinear_nonlinear)
        t+=1
    return linear_solutions

# Solutions of nonlinear term ---- gives array of length (Total_time*fps)
def nonlinear_linear(theta_initial,Total_time,fps):
    nonlinear_solutions = zeros([Total_time*fps+2])
    nonlinear_solutions[0] = theta_initial
    phi = zeros([Total_time*fps+2])
    phi[0],t= 0,0
    while t<Total_time*fps:
        nonlinear_solutions[t+1], phi[t+1] = RK4(nonlinear_solutions[t],phi
```

```
            [t],1/fps,f2nonlinear_linear)
        t+=1
    return nonlinear_solutions

def nonlinear_nonlinear(theta_initial,Total_time,fps):
    nonlinear_solutions = zeros([Total_time*fps+2])
    nonlinear_solutions[0] = theta_initial
    phi = zeros([Total_time*fps+2])
    phi[0],t= 0,0
    while t<Total_time*fps:
    nonlinear_solutions[t+1], phi[t+1] = RK4(nonlinear_solutions[t],phi[t
        ],1/fps,f2nonlinear_nonlinear)
    t+=1
    return nonlinear_solutions

# ------------(for graphs)----------
# this describes frequncy of nonlinear term.
def w_nonliner(theta_initial):
w_ = (sqrt(l/g))*(1+(0.25*(sin(0.5*theta_initial))**2)+((9/64)*(sin(
    theta_initial*0.5))**4))
return 1/w_

# phase plane defination
def linear_phase_plane(theta,phi):
    f1 = phi
    f2 = -((gammal)*phi)-(w0*theta)
    return f1,f2

def nonlinear_phase_plane(theta,phi):
    f1 = phi
    f2 = -((gammal)*phi)-(w0*sin(theta))
    return f1,f2

def linear_phase_planeq(theta,phi):
    f1 = phi
    f2 = -((gammaq)*phi*phi)-(w0*theta)
    return f1,f2

def nonlinear_phase_planeq(theta,phi):
    f1 = phi
    f2 = -((gammaq)*phi*phi)-(w0*sin(theta))
    return f1,f2



# ---------------------------------
```

Simulation files are *lindamsamepos.py, nonlindamsamepos.py* and also two other *lindamdiffpos.py* and *nonlindamdiffpos.py*. In the files respecevely i put

my code for linear damping with same pivot position and also quadratic damping with same position of pivot. Other two files are complementary for different pivot position not much of importance.

I am writing here my linear damping file for regeneration perpose in other two i just changed fuction name from *main.py*.

```
from main import *
import pygame
import time
from numpy import cos, sin, pi


# this part done.
# just do - [ ] dual color balls



def bob1(window,x,y):
    image = pygame.image.load("bitmap1.png")
    window.blit(image, (x,y))

def bob2(window,x,y):
    image = pygame.image.load("bitmap2.png")
    window.blit(image, (x,y))

def surface(window,x,y):
   image = pygame.image.load("surface.png")
   window.blit(image, (x, y))



def position(l,theta):
   return origin_x-10+l*cos((1.5*pi)-theta),origin_y-10-l*sin((1.5*pi)-
       theta)

def mainloop(window,fps):
    global t
    run = True
    clock = pygame.time.Clock()
    c = 0
    while run:
        for event in pygame.event.get():
            if event.type== pygame.QUIT:
                run= False
                break
        clock.tick(fps)
```

```
        x,y = position(l,exact[c])
        x2,y2 = position(l,appro[c])
        if c==len(exact):
            break
        window.fill("#ffffff")
        surface(window,200,origin_y-15)
        pygame.draw.aaline(window,color="#5BDEC1",start_pos=(origin_x,
            origin_y),end_pos=(x+radius,y+radius))
        pygame.draw.aaline(window,color="#5BDEC1",start_pos=(origin_x,
            origin_y),end_pos=(x2+10,y2+10))
        bob1(window,x,y)
        bob2(window,x2,y2)
        pygame.display.update()
        c+=1
        pygame.quit()

 if __name__ == "__main__":
    pygame.init()
    window = pygame.display.set_mode((width,height))
    exact = nonlinear_linear(theta_initial,300,fps)
    appro = linear_linear(theta_initial,300,fps)
    t = time.perf_counter()
    mainloop(window, fps)
```

4.3 (Appendix:3) Code for graphs

This is code for all the graphs in this document. I have written two functions which gives one plot for angular frequancy and other gives θ vs time t for different θ_0.

```
   from main import *
import numpy as np
import matplotlib.pyplot as plt




def angfre():
    A = np.linspace(-np.pi,np.pi,6000)
    w = np.zeros(np.size(A))
    w0 = np.zeros(np.size(A))

    for i in range(np.size(A)):
        w[i] = w_nonliner(A[i])
        w0[i] = np.sqrt(g/l)


    plt.figure()
```

```
    plt.plot(A,w, label="angular frequency of nonlinear solution")
    plt.plot(A,w0, label="angular frequency of linear solution")
    plt.title("angular frequency")
    plt.ylabel("angular frequency")
    plt.xlabel("initial displacement")
    plt.legend()
    plt.savefig("angfre.png")
    plt.close

def thetawitht(total_time):
    fig = plt.figure()
    # fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(2, 2, sharex=True, sharey=
        True)
    gs = fig.add_gridspec(2, 2)
    (ax1, ax2), (ax3, ax4) = gs.subplots()

    fig.suptitle('Angular displacement vs time t')
    # plt.xlabel("time")
    # plt.ylabel("angular displacement")

    time = np.linspace(0,total_time,1002)



    theta_initial = pi/10
    soll = linear_linear(theta_initial,total_time,int(1000/total_time))
    soln = nonlinear_linear(theta_initial,total_time,int(1000/total_time))
    ax1.plot(time,soll)
    ax1.plot(time,soln)
    ax1.set_title("theta = pi/10")
    # plt.ylabel("angular displacement")


    theta_initial = theta_initial*2
    soll = linear_linear(theta_initial,total_time,int(1000/total_time))
    soln = nonlinear_linear(theta_initial,total_time,int(1000/total_time))
    ax2.plot(time,soll)
    ax2.plot(time,soln)
    ax2.set_title("theta = pi/5")

    theta_initial = pi/2
    soll = linear_linear(theta_initial,total_time,int(1000/total_time))
    soln = nonlinear_linear(theta_initial,total_time,int(1000/total_time))
    ax3.plot(time,soll)
    ax3.plot(time,soln)
    ax3.set_title("theta = pi/2")

    theta_initial = pi
    soll = linear_linear(theta_initial,total_time,int(1000/total_time))
```

```
    soln = nonlinear_linear(theta_initial,total_time,int(1000/total_time))
    ax4.plot(time,soll)
    ax4.plot(time,soln)
    ax4.set_title("theta = pi")
    fig.tight_layout()
    plt.savefig("thetawitht.png")
    plt.close()


angfre()
thetawitht(40)
```

And this is the code that i written for plotting phase planes,

```
    from numpy import linspace, meshgrid
from main import *
import matplotlib.pyplot as plt


def exact_lineardamp():
    x1 = linspace(-6,6,2000)
    x2 = linspace(-6,6,2000)

    x1_,x2_ = meshgrid(x1,x2)

    u,v = nonlinear_phase_plane(x1_,x2_)
    vel = sqrt(u**2+v**2)

    plt.figure()
    plt.streamplot(x1_,x2_,u,v, color='k', linewidth=0.8,density=1.5,
        minlength=0.01, arrowsize=0.8,arrowstyle="->")
    plt.title("stream plot of equation without approximation")
    plt.xlabel("$\theta$")
    plt.ylabel("$\phi$")
    plt.savefig("exactlstr.png")

def appro_lineardamp():
    x1 = linspace(-6,6,2000)
    x2 = linspace(-6,6,2000)

    x1_,x2_ = meshgrid(x1,x2)

    u,v = linear_phase_plane(x1_,x2_)
    vel = sqrt(u**2+v**2)

    plt.figure()
    plt.streamplot(x1_,x2_,u,v, color='k', linewidth=0.8,density=1.5,
        minlength=0.01, arrowsize=0.8,arrowstyle="->")
    plt.title("stream plot of approximated equation")
    plt.xlabel("$\theta$")
```

```
    plt.ylabel("$\phi$")
    plt.savefig("approlstr.png")

exact_lineardamp()
appro_lineardamp()
```

References

[1] Richard Bronson and Gabriel B Costa. *Schaum's outline of differential equations*. McGraw-Hill Education, 2014.

[2] Herbert Goldstein, Charles Poole, and John Safko. Classical mechanics, 2002.

[3] Walter RA Goossens. Review of the empirical correlations for the drag coefficient of rigid spheres. *Powder Technology*, 352:350–359, 2019.

[4] Kim Johannessen. An analytical solution to the equation of motion for the damped nonlinear pendulum. *European Journal of Physics*, 35(3):035014, 2014.

[5] Dominic Jordan and Peter Smith. *Nonlinear ordinary differential equations: an introduction for scientists and engineers*. OUP Oxford, 2007.

[6] Marko V Lubarda and Vlado A Lubarda. An analysis of pendulum motion in the presence of quadratic and linear drag. *Eur. J. Phys*, 42(055014):055014, 2021.

[7] VD Majumdar and MB Vajifdar. Coefficient of viscosity of air. In *Proceedings of the Indian Academy of Sciences-Section A*, volume 8, pages 171–178. Springer, 1938.

[8] Robert Salamon, Henryk Kamiński, and Paweł Fritzkowski. Estimation of parameters of various damping models in planar motion of a pendulum. *Meccanica*, 55(9):1655–1677, 2020.

[9] MG Sobamowo. Exact analytical solutions of nonlinear differential equation of a large amplitude simple pendulum. *World Scientific News*, 144:70–88, 2020.

[10] JC UPADHYAY. *CLASSICAL MECHANICS.* Himalaya Publishing House, 2016.