DAMPING OSCILLATION WITH PYTHON

Course code: SPHYPR03

Course: Mathematical Physics (SPHY0302)

Month and year of submission: October 2020

By,

Pratham Doiphode   Roll no: 431   UID: 192625

Rajapandi Nadar   Roll no: 423   UID: 192585

Abstract:

The main objective of the project is to visualize the damping oscillation by solving the second order differential equation using numerical method (Euler’s method) using Pycharm software (a Python IDE). In this project we have used the second order differential equation of damping oscillation and solved it with the help of the Euler’s method of numerical integration, and with the help of the Vpython we have created a model of the spring mass oscillator and by creating a loop where we will use the solution of the differential equation to assign the bob values such as positions and velocity so that we can witness a oscillatory motion of the system. Further we will change some physical parameter of the spring such as damping constant and force constant and see the changes in the oscillatory motion of the system. Finally will plot ‘**time vs displacement’** graph for the clear understanding of the Damping oscillation.

INDEX PAGE

Introduction:

The main aim of the project is to solve a second order differential equation of damping oscillation using numerical integration method, and plotting the damping oscillation graph [time vs displacement]. Will also create a spring mass system using VPython and finally with the help of the solution of the differential equation we control the oscillatory motion of the system, and observe the behavior of the system in different conditions.

Background:

Ordinary Differential equation:

A differential equation is an equation that relates one or more function and their derivatives, usually the functions represent some physical quantities ‘like if x is the displacement then the first derivative will be its velocity’. The ordinary differential equation (ODE) is a differential equation containing one or more function of independent variable and the derivatives of that function and the term ordinary is used in contrast with partial differential equation which may be with respect to more than one independent variable. The general solution of the ODE will be an n^{th} order equation and a particular solution by applying boundary conditions or initial conditions.

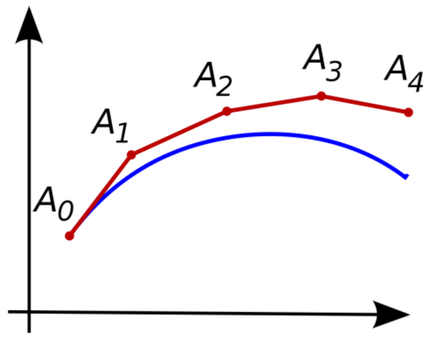
Damping oscillation:

An oscillatory motion whose amplitude dies out with is known as a damping oscillatory motion and the process is called as the Damping oscillation. The damping has been classified into three parts namely critical damping, over damping and under damping. In our day to day life we may have witnessed many damping oscillation phenomenon, some of the examples are guitar strings, and the suspensions of the vehicle and many more.

<u>Theory</u>:

Euler method for solving the differential equation.

Euler method is a first order numerical procedure for solving ordinary differential equation with a given initial value since it a method for solving the first order method hence the error is directly proportional to the step size means as we decrease the step size the error in the solution also decreases.



As shown in the figure, the blue curve is the actual solution of the graph with initial condition as A_o and the red curve is the solution that we have obtained with the help of the Euler method & we can clearly say that the obtained solution will match the actual solution if we decrease the step size between A_0 and A_1.

Since Euler method cannot be used to solve differential equation of higher order, for solving the differential equation of the second order we will break the differential equation of second order into two first order differential equations.

For example: Consider a second order differential equation

$$x`` = f(t,x,x`) \;\; ; \;\; x(t_o) = x_o \;\; ; \;\; x`(t_o) = u_0$$

First differential equation $x` = u$

Second differential equation $u` = f(t,x,u)$

<u>Experimental</u>:

The flow of the experiment is such that we will first import the Vpython module in the pycharm software (Version: 2020.2.3) and will design the spring mass oscillator and then we will use the second order differential equation of the damping oscillation for obtaining the position and velocity of the bob.

Flow of the code:

1. Installing the required software such as python, Vpython, Matplotlib and other essential modules.
2. Creating the different components in the spring mass system such as bob, rigid support and the spring by using the Vpython module.
3. Then selecting the differential equation for the system (x`` = (k/m)*x – (b/m)*x`).
4. Breaking the second order differential equation into two different first order differential equations.
5. First ODE x = x` ;  second ODE  x` = (k/m)*x – (b/m)*x`.
6. By the Euler method of numerical integration, we will solve both the differential equation simultaneously.
7. Will define the array for storing the instantaneous values of the velocity, time and the displacement.
8. Will assign the instantaneous values to the bob's attributes like velocity and position as time varies.
9. Will take input command from the user for getting the spring attributes such as damping and the force constant as well as bob attributes like position of the bob.
10. And finally will write code for plotting the time vs displacement graph.

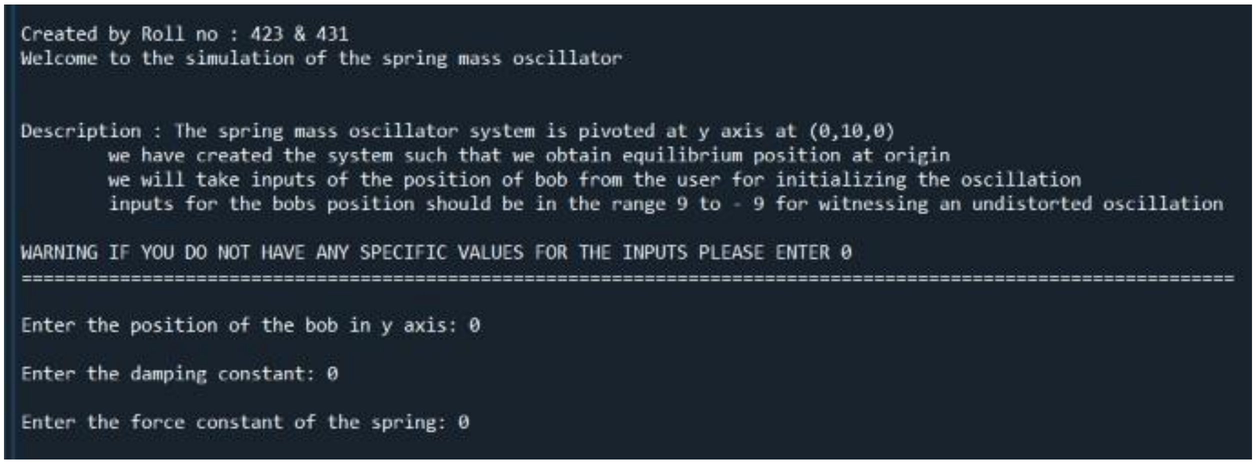
It is up to an individual which type of loops to be used for this program and also there are many ways to assign the instantaneous values to an array.

CODE:

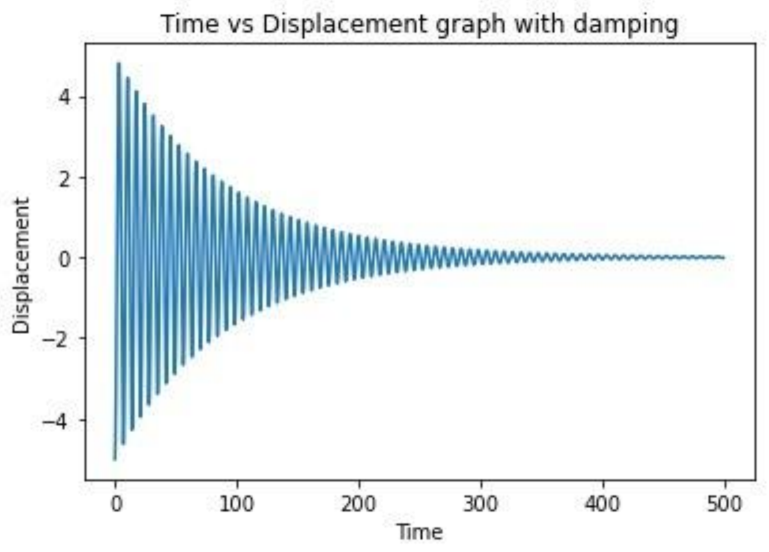
```python
#Python coding
#Code by Rajapandi & Pratham
#Roll no: 423 & 431
#This is the python code for visualizing the damped oscillation by solving the differential equation as a base


#Idea of the System
#   The Project is to visualize a famous physical concept "THE DAMPING OSCILLATION"
#   Here in this system the rigid support is on y axis at (0,10,0)
#   A spring is attached to it and a cube(bob) is attached at the other end of the spring
#   We have designed a default system in case the user is new to this concept


from vpython import*
import matplotlib.pyplot as ra


#Function func is defined such that it solves he functional part while solving the differential equation
#This function takes 5 inputs and returns d after completing the operation
def func(k,m,b,x,v):
    d = (k/m)*x-(b/m)*v
    return d
def fun(k,m,b,x,v):
    d = -(k/m)*x - (b/m)*v
    return d
#Here we have displayed some of the instruction for visualizing the system
#Major instructions are regarding the input values that an user should input
print()
print('Created by Roll no : 423 & 431')
print('Welcome to the simulation of the spring mass oscillator')
print()
print()
print('Description : The spring mass oscillator system is pivoted at y axis at (0,10,0)')
print('        we have created the system such that we obtain
equilibrium position at origin')
print('        we will take inputs of the position of bob from
the user for initializing the oscillation ')
print('        inputs for the bobs position should be in the
range 9 to - 9 for witnessing an undistorted oscillation ')
print()
print('WARNING IF YOU DO NOT HAVE ANY SPECIFIC VALUES FOR THE
INPUTS PLEASE ENTER 0 ')
print('=================================================================
======================================================')


# Main part of the code which is used to control the motion of
the bob
while True:      #  While True loop is used for continuously
asking inputs from user if the input value is not in the desired
form

    #   try and expect is used in order to control the value
error that the user may come across
    try:

    #   Taking input from the user for the position of the bob ,
damping constant and force constant of the spring
        x = float(input('Enter the position of the bob in y
axis: '))
        b = float(input('Enter the damping constant: '))
        k = float(input('Enter the force constant of the spring:
'))




    #   Creating the default values for each of the inputs taken
by the user this will be executed if the input entered is 0
        if x == 0:
            x=-5
        if b == 0:
            b = 0.25
        if k == 0:
            k = 8


    # Defining three arrays for collecting data for plotting the
graph of time vs displacement
        y = [x]
        t = [0]
        v = [0]


    #   Defining time interval of the oscillation
        ti = 0
        tf = 500
        n = 150000
        dt = (tf - ti) / n

    #   Defining the system with the appropriate dimensions
        eq=vector(0,0,0)
    #   Wall is the rigid support in the system

wall=box(pos=vector(0,10,0),size=vector(3,0.3,2),color=color.whi
te)
    #   bob with all the attributes like position, mass and
velocity

bob=box(pos=vector(0,x,0),velocity=vector(0,0,0),size=vector(1,1
,1),mass=10.0,color=color.blue)
    #   pivot is defined for the fixed end of the spring
        pivot=vector(0,10,0)
    #   spring with all the attributes such as force constant &
damping constant
        spring=helix(pos=pivot,axis=bob.pos-
pivot,radius=0.4,constant=k,thickness=0.1,coils=15,dampingconst=
b,color=color.gray(0.5))

    # This part is for plotting the graph of the damping
oscillation along side the model to get a great connect between
the graph and the model
        j = 1
        while j < n:
            v1 = v[j-1] + dt * fun(k,10,b,y[j-1],v[j-1])
            v.append(v1)
            y1 = y[j-1] + dt*v[j-1]
            y.append(y1)
            t1 = t[j-1] + dt
            t.append(t1)
            j += 1

    # Plotting the graph
        ra.plot(t, y)
        ra.xlabel('Time')
        ra.ylabel('Displacement')
        ra.title('Time vs Displacement graph with damping')
        ra.show()


    #   This part of the code is for solving the second order
differential equation  by euler's method
    #   The solution is obtained here by breaking second order
differential equation into two different first order
differential equations
    #   One of the differential equation is (x'[i] = x'[i-1] +
dt*function)
    #   The second differential equation is (x[i] = x[i-1] +
dt*function)
        i=1
        while i < n:
            rate(500)
            bob.velocity = bob.velocity +
dt*func(spring.constant,bob.mass,spring.dampingconst,eq-
bob.pos,bob.velocity)
            bob.pos = bob.pos + dt * bob.velocity
            y.append(bob.pos)
            spring.axis = bob.pos - spring.pos
            i = i+1




    except ValueError:
        print('Only numbers are allowed....')
        print('Please enter only numbers....')
        print('Try again')
        print('--------------------------------------------------
------------------------------------------------------')
    else:
        break
```

Results:

The spring mass oscillator system is pivoted at y axis at (0,10,0). We have created the system such that we obtain equilibrium position at the origin. The bob's position should be in the range +9 to -9 for witnessing an undistorted oscillation. When we run the above program, we get the model of spring mass oscillator. Also, we are asked to enter the position of the bob in y axis, enter the damping constant & finally enter the force constant of the spring. If we don't have any specific values, we can enter zero. We also get the graph of time vs displacement with damping. More precisely, we can say light damping.



```
Created by Roll no : 423 & 431
Welcome to the simulation of the spring mass oscillator


Description : The spring mass oscillator system is pivoted at y axis at (0,10,0)
        we have created the system such that we obtain equilibrium position at origin
        we will take inputs of the position of bob from the user for initializing the oscillation
        inputs for the bobs position should be in the range 9 to - 9 for witnessing an undistorted oscillation

WARNING IF YOU DO NOT HAVE ANY SPECIFIC VALUES FOR THE INPUTS PLEASE ENTER 0
==============================================================================================================

Enter the position of the bob in y axis: 0

Enter the damping constant: 0

Enter the force constant of the spring: 0
```

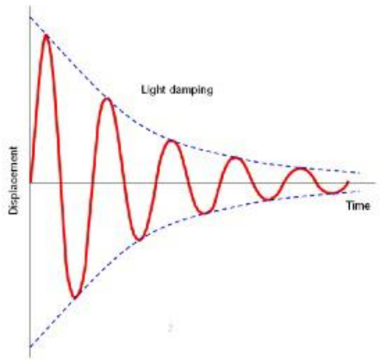


Applications:

The above program can be used for explaining the concept of damping oscillation. There are basically three different of dampings namely light damping, critical damping & heavy damping.

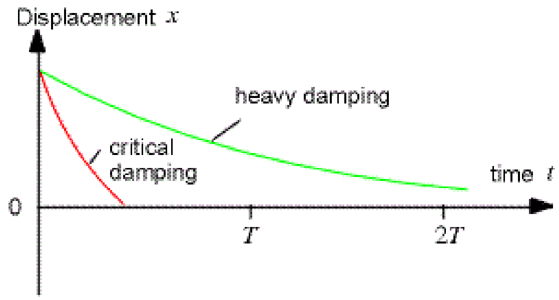
1. Light Damping:

Defined oscillations are observed, but the amplitude of oscillation is reduced gradually with time.



2. Critical Damping:

The system returns to its equilibrium position in the shortest possible time without any oscillation.



3. Heavy Damping:

The system returns to the equilibrium position very slowly, without any oscillation. Heavy damping occurs when the resistive forces exceed those of critical damping.

<u>Summary:</u>

We visualized the damping oscillation by solving the second order differential equation using numerical method (Euler's method) using Pycharm software (a Python IDE). We used the second order differential equation of damping oscillation and solved it with the help of the Euler's method of numerical integration, and with the help of the Vpython we created a model of the spring mass oscillator and by creating a loop where we used the solution of the differential equation to assign the bob values such as positions and velocity. Then we changed some physical parameter of the spring such as damping constant and force constant. Finally, we got a plot of **'time vs displacement'** for the clear understanding of the Damping oscillation.

References:

1. We reffered seventh edition of the book "Numerical Methods for Engineers" by Steven C. Chapra & Raymond P. Canale for understanding the concept of Euler's method more clearly.

2. We also reffered fifth edition of the book "A Primer on Scientific Programming with Python" by Hans Petter Langtangen for gaining some knowledge about how to solve differential equations using python.

3. We took the help from Wikipedia pages for the sake of definitions & also took images from google for the betterment of our project report.

Acknowledgement:

We would thank all the faculties of Physics Department for guiding us throughout the project. Also, we are very thankful to our teacher incharge, Prof Radhekrishna Dubey sir for always cooperating us.

The End

THANKYOU!