Adrian Loekman
UID: 105785820
MAE 82
December 15, 2021

Forced Van der Pol Oscillator

Deliverable 1 (Python)

```
# Import necessary packages
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate as si
# set a and b
a, b = 0, 300
# create the timespan
t = np.linspace(a, b, 10000)
F = 0.2239  # mu equals epsilon and F
omega = (2*math.pi)/10  # set omega as (2*pi)/T
epsilon = 1.8*math.pi

# create a  def to define functions for the second order linear equation
def vdp_driven(t, z):
    x, y = z  # set the variables of the system of ODE
    return [y, epsilon/omega * (1 - x ** 2) * y - x +
epsilon*F*math.cos(omega*t)]
    # define the second order ODE as a 2  row vector

sol = si.solve_ivp(vdp_driven, [a, b], [1, 0], t_eval=t)  # package to solve
Initial Value Problems for ODE
plt.grid(visible=None, which='major', axis='both', color='k', linestyle='-',
linewidth=0.5)
plt.plot(sol.y[0], sol.y[1])  # plot the solutions
# Plot Details
plt.xticks(np.arange(-2.5, 2.5+1, step=0.5))
plt.yticks(np.arange(-20, 20+1, step=5))
plt.xlim(-2.5, 2.5)
plt.ylim(-20.20)
plt.title("Phase Plane Plot of a Particular Solution")
plt.xlabel("x")
plt.ylabel("dx/dt")
plt.show()  # make plot visible
```

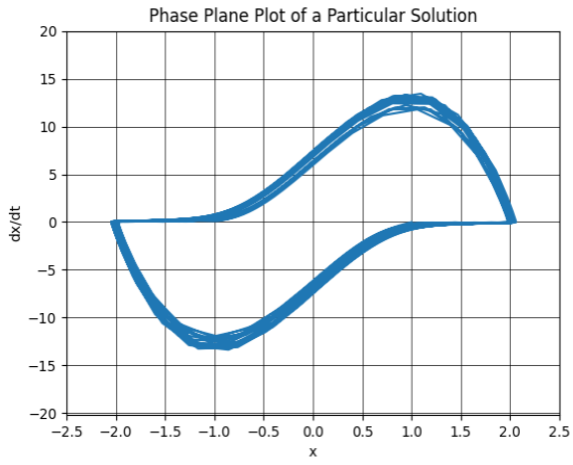

Figure 1 Phase Plane Solution using Python

Adrian Loekman
UID: 105785820
MAE 82
December 15, 2021

Deliverable 1 (MATLAB)

```
%% Set Parameters for differential equation
F = 0.2239;
omega = (2*pi)/10;
epsilon = 1.8*pi;

%% Solve Differential Equation
fun = @(t,x) [x(2);(epsilon./omega).*(1-(x(1).^2)).*x(2) - x(1) +
epsilon.*F.*cos(omega.*t)];
[t1,x1] = ode45(fun,[0,300],[1,0]);

%% Plot Phase Plane
plot(x1(:,1),x1(:,2))
title("Phase Plane Plot of a Particular Solution")
xlabel("x")
ylabel("dx/dt")
grid on
```

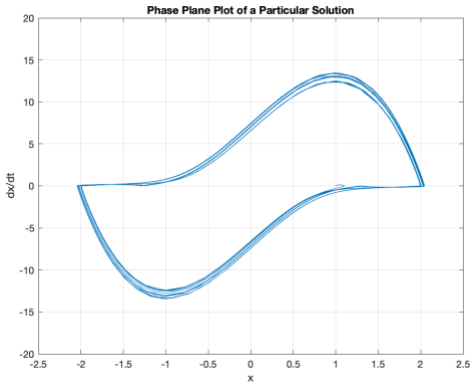

Figure 2 Phase Plane Solution using MATLAB

Adrian Loekman
UID: 105785820
MAE 82
December 15, 2021

Deliverable 2 (MATLAB)

```
clc; clear all; close all;
%% Set Parameters
F = 0.2239;
omega = (2*pi)/10;

%% Solve and Plot for epsilon [1,5]
for epsilon = 1:0.01:5
    fun = @(t,x) [x(2);(epsilon./omega).*(1-(x(1).^2)).*x(2) - x(1) +
epsilon.*F.*cos(omega.*t)];
    [t1,x1] = ode45(fun,0:5:300,[0,1]);
    eps = epsilon*ones(length(x1),1);
    plot(eps,x1(:,1),".","markersize",2)
    hold on
end
title(' \epsilon = [1, 5] ')
xlabel('\epsilon')
ylabel('x')
hold off

%% Solve and Plot for epsilon [4.65,4.95]
figure(2)
for v = 4.65:0.0005:4.95
    fun2 = @(s,y) [y(2);(v./omega).*(1-(y(1).^2)).*y(2) - y(1) +
v.*F.*cos(omega.*s)];
    [t2,x2] = ode45(fun2,0:5:300,[1,0]);
    eps2 = v*ones(length(x2),1);
    plot(eps2,x2(:,1),".","markersize",2)
   hold on
end
title(' \epsilon = [4.65, 4.95] ')
xlabel('\epsilon')
ylabel('x')
```

Adrian Loekman
UID: 105785820
MAE 82
December 15, 2021

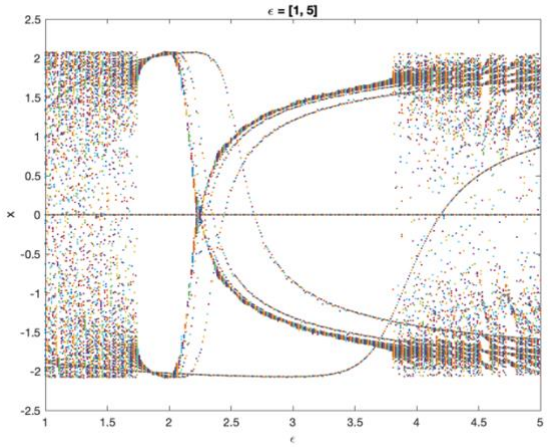

Figure 3 Bifurcation Diagram when epsilon = [1,5]

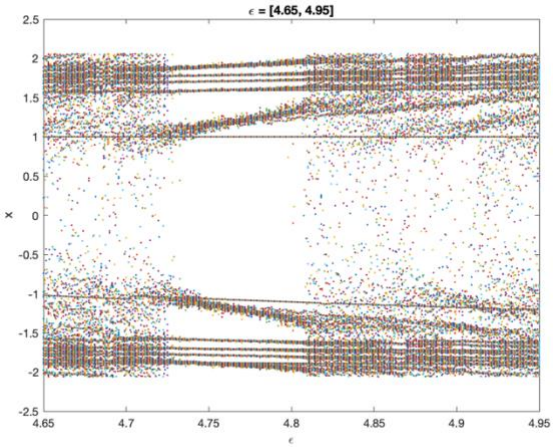

Figure 4 Bifurcation Diagram when epsilon = [4.65,4.95]

Deliverable 3

For deliverable 1, the most important part is to solve the second order differential equation given on the problem statement. When working with Python, import all necessary packages onto the script. Then, create a timespan for the computer to solve the differential equation. In this case, I will use t = 0 to 300 with 10000 timesteps. Also, define every parameter that are useful for this equation such as F, omega, and epsilon. Next, set up a function that will be used when applying the ODE solver. Since the problem is a second order ODE, we need to split them into two first order ODEs. Solve the ODEs using a Python ODE solver using the time span created and the initial conditions 1 and 0. Finally, plot the values of x and dx/dt. This is gotten from the 1st solution and 2nd solution respectively.

In MATLAB, a similar process is used. First, define all parameters (F, epsilon, omega) and set values for them. Then, create a function with a function handle t,x which will be solved using the ode solver. Then, solve the ODE using ode45. The result of x will come in two columns, the first is the values of x (from the integration of the first derivative) and the second is dx/dt (from the integration of the second derivative). Plot the values of x in the x axis and dx/dt in the y axis. The plot will look like some sort of circular figure that goes through a loop. I believe this may be the result of having a cosine in the function. Otherwise, it would just go once instead of a loop.

The bifurcation diagram is more challenging to do. The main goal is to plot the x values for every varying epsilon. In order to do this, define the known parameters, F and omega, with its respective values. Then, create a for loop to solve the ODE for each value of epsilon, then plot it onto the figure. Do this for all values of epsilon desired. The bifurcation diagram tells us that chaotic and periodic motion happens in this scenario. Inside the loop, we have the function with the function handle. Then, solve it using ode45 with its time span and initial conditions. The time span used here is specified (i.e., we set the number of time elements). Then, plot the epsilon with the x values in the first column. The reason we use hold on command is to make sure that the next plot is also plotted in the same figure. This way, we can see how each epsilon has different values for x.