CS5016: Computational Methods and Applications

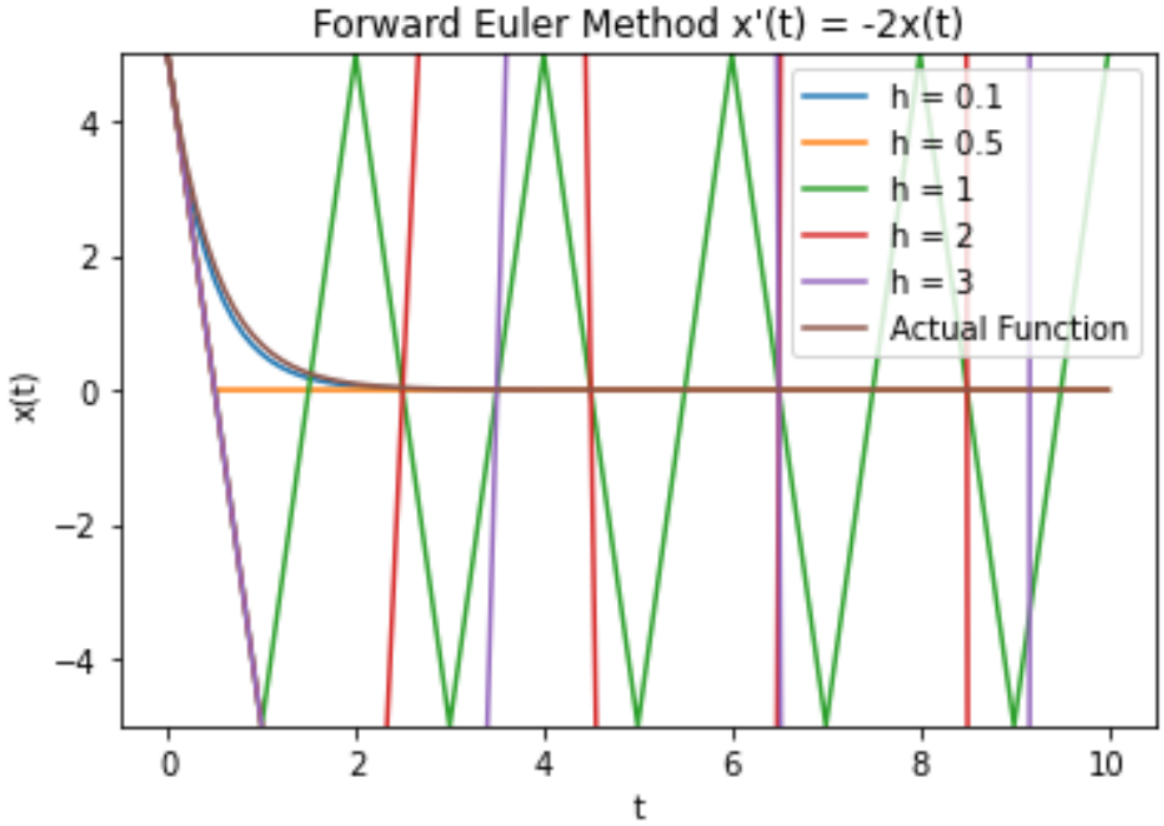
Lab Report - 6

111901030
Mayank Singla

Q1.

I am creating the function `solveODE` to solve the given problem using the forward Euler method. I am first creating a nested function `get_points` to get all the t_n points in the given range. Then for each input step size, using a for loop I am evaluating the x_n points for each step size and plotted all the curves on the same graph. I am then also plotting the actual graph by taking the function as input. I am making two lambda functions `inp_func` and `inp_ode` to evaluate the values of the given function which I solved by hand and the given ODE in the question, which I am then passing as arguments to the `solveODE` function.

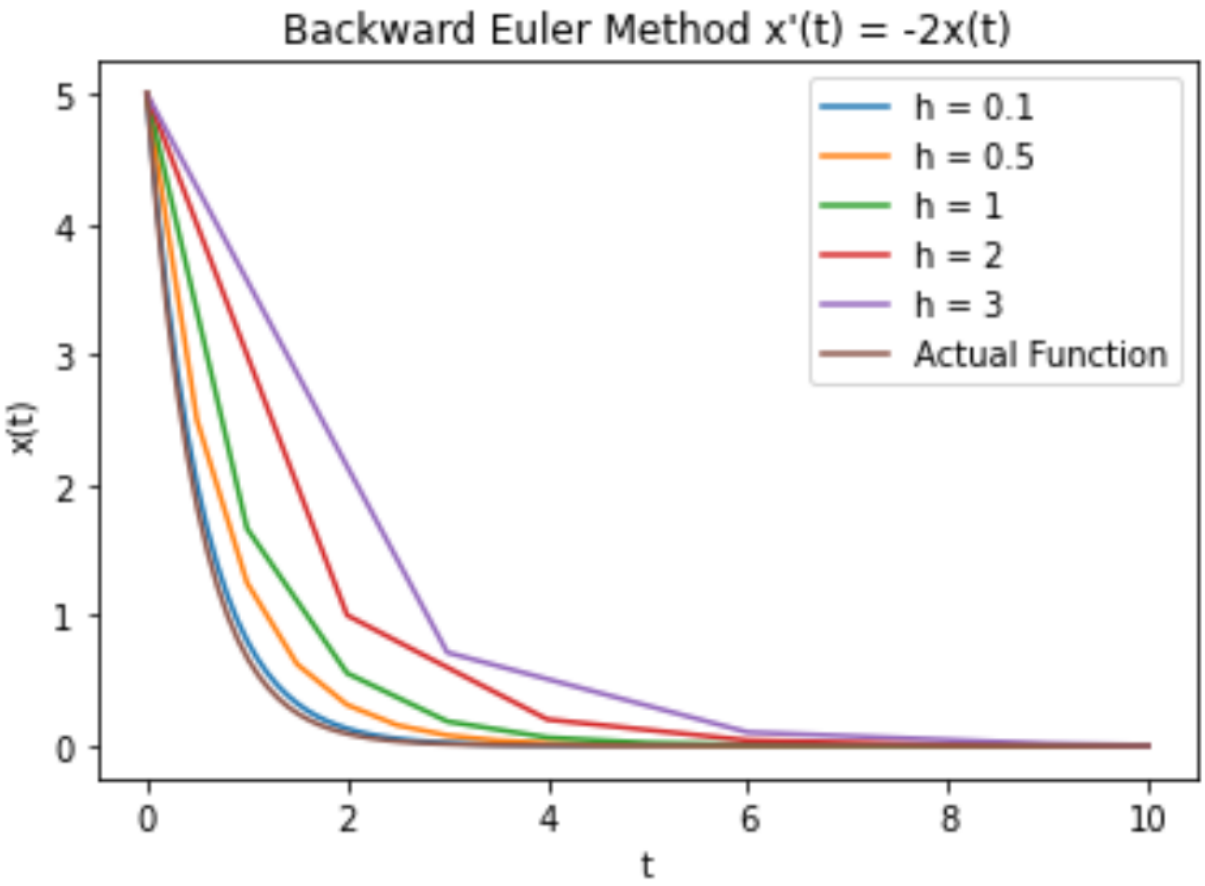
Here is the output I obtained for the given values in the question.



Q2.

I am again doing the same things as in the previous question just here I am using the Backward Euler method to solve the same given ODE and thus only changing my `inp_ode` lambda function. Here as it was just a simple ODE, I was able to obtain an expression for x_{n+1} simply in terms of x_n and evaluate the values.

Here is the output I obtained for the given values in the question.



Q3.

Here first of all I was trying to solve the given system of ODEs by converting it into a matrix form, but due to the `sinΘ` term I was unable to do so, so I initially assumed that I could take `sinΘ` is approximately equal to `Θ` which allows reducing the given system of ODEs into the matrix form as

```
x′ = |   0       1   | x
     | (-g/L)    0   |
```

where $x = \begin{vmatrix} \Theta \\ v \end{vmatrix}$

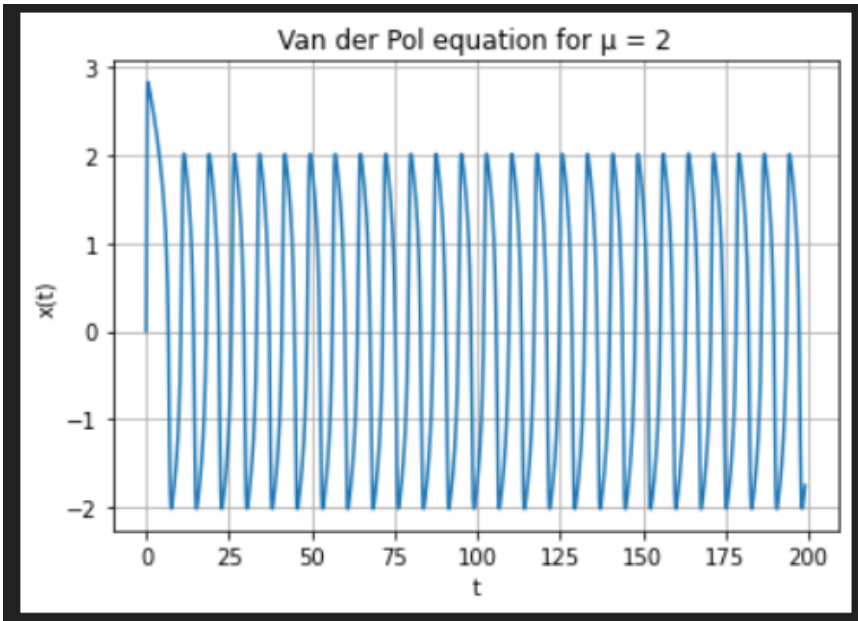
Though I was able to get to the solution, the results were not good for large Θ which was also expected. Then I realized that being only just two equations, I need not convert them into the matrix form and I could have simply solved the equations and evaluated both the points using a simple for loop only, which I did in the code as well.
I simply evaluated the different values of theta at different time intervals and then plotted them using the `matplotlib` animation. I used normal two points plot to plot a line and `plt.Circle` to plot the circular bob and created all the required functions for the `FuncAnimation` and simulated the simple gravity pendulum. I observed that to get the convergence, we need to choose a smaller step size to see it.
We could check the simulation of it by running the code as it is not possible to output it here. I have attached the gif of my output for the following initial values.

```
# Testing the function
ani = solveODE(
    ode=inp_ode, theta0=math.pi / 4, v0=0, h=0.001, t0=0, T=10, g=10, L=0.1
)
```

Q4.

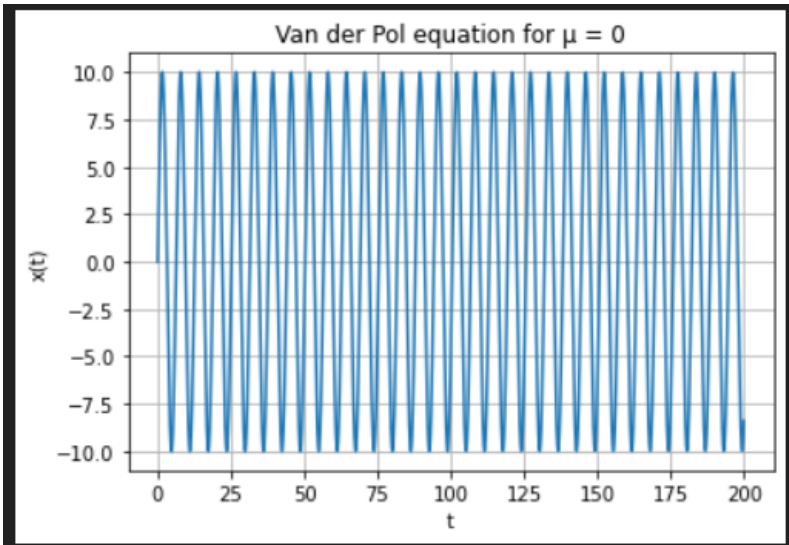
I am creating the function `solveODE` to solve the given problem using the `scipy.integrate.solve_ivp` function. This function is taking the value of `μ` and all the other initial values and then I am generating some random values for the time using `np.linspace` and solving the required system of ordinary differential equations. To `solve_ivp` function, I need to provide a function which I named `vdp_derivatives` which returns the derivatives of the system of ODEs by reducing the given ODE to two ODEs for order one. Then from the solution, I am extracting the values of `x` points that I obtained and plotted them on the graph. To evaluate the period I am taking two crossings of the curve from the last and taking the time difference between them as the time interval. A crossing is whenever the curve changes its sign and the difference between the time values of two such points is the Period for the given function.
Here is an example output I obtained.

```
# Testing the function
solveODE(x0=0, v0=10, mu=2, t0=0, T=199, n=10000)
```




```
The time period of the curve for μ = 2 is 7.62
```

```
# Testing the function
solveODE(x0=0, v0=10, mu=0, t0=0, T=200, n=10000)
```




```
The time period of the curve for μ = 0 is 6.30
```

Q5.

Here again, as I did in the previous question I am solving the given 2nd order ODE by reducing it to two first-order ODEs and then using the `scipy.integrate.solve_ivp` function to solve the given system of ODEs. The only difference here is that we are given three 2nd order ODEs and each r_i has two components which leads us to have a total of 12 variables to solve simultaneously. I am solving them all using the same `vdp_derivatives` function that was required by the `solve_ivp` function to return the derivatives of the system of differential equations. Then from the solution obtained, I extracted all the coordinates for the three bodies for each particular time and then plotted them all using the `FuncAnimation` function and for that made all the required utility functions as in the 3rd question.

In this question, I initially encountered a problem where for some points the norm of the vector difference was becoming 0 due to which the points were going to infinity as the norm is in the denominator. I fixed this problem by taking a dummy value for the norm whenever it becomes zero.

We could check the simulation of it by running the code as it is not possible to output it here. I have attached the gif of my output for the following initial values.

```
# Testing the function
r10 = [0, 0]
r20 = [3, 1.73]
r30 = [3, -1.73]
v10 = [0, 0]
v20 = [0, 0]
v30 = [0, 0]
ani = solveODE(
    init_r=[*r10, *r20, *r30], init_v=[*v10, *v20, *v30], t0=0, T=400, n=1000
)
```