CS102, Python, Practical 7, Python Classes

Please email your solution to Question 1 as an attachment to the course lecturer at the following address:

martin.meere@nuigalway.ie

Please note the following:

(a) **your program should be saved as**
motion_yourfirstname_yourlastname.py;

(b) **as a backup, put your name and student id number in a comment at the head of the program;**

(c) **the deadline for submission is April the 9th;**

(d) **programs that do not work at all will receive 0%;**

(e) **include sufficient comments for the program to be comprehensible;**

(f) **you may get assistance from the tutors (I encourage you to do so in fact) or the lecturer, but people who simply copy or share work risk losing some or all of their marks for this practical.**

1. *One_d_motion Class.* On pages 130 to 137 of your notes, an algorithm for tracking the one-dimensional motion of a particle is described. Write a `One_d_motion` class to model the one-dimensional motion of a particle, and save it in a module named `motion.py`; this should be quite similar to the `Planet` class of the previous practical. Your class should contain the following methods.

   `__init__(self, pos, vel)` The constructor. Here `pos` and `vel` give the position and velocity, respectively, of the particle at a given moment. One noteworthy difference with the `Planet` class is that you should probably include an instance variable for the time, `self.t`, say, and initialize it to -20.0, that is, include a statement of the kind `self.t = -20.0` in the constructor.

   `get_pos(self)` Accessor method that returns the position of the planet. Similar accessor methods should be included for `vel` and `t`.

   `update(self,dt)` This method updates the position and velocity of the particle for a time `dt` later; the appropriate algorithm is given on page 132

of your notes. For example, to update `vel`, a command of the following kind can be used:

$$\texttt{vel\_new} = \texttt{self.vel} + \texttt{dt} * \texttt{self.\_\_f}(\texttt{self.pos}, \texttt{self.vel}, \texttt{self.t})$$

where `__f` is defined below. Remember to update the time as well. Once ALL of the new (updated) quantities have been calculated, you can then update the instance variables using commands of the following kind

$$\texttt{self.vel} = \texttt{vel\_new}$$

`__f(self,x,v,t)` Helper function that returns the force - see page 133 of your notes. Here `x,v,t` refer to the position, velocity and time, respectively. When you write this function first, just get it to return `-x`; this models an undamped, unforced oscillator. To model other forces, we just need to modify `__f` appropriately.

Include a docstring for the class, and one for each of the methods.

2. You have been emailed a program called plot_motion.py which plots the one-dimensional motion of a particle. The program needs the `One_d_motion` class of Question 1 to run, and so you will need to write this class before trying to run the program. The plotter should provide a useful check on your `One_d_motion` class.

   The program is set up to plot position versus velocity, that is, the motion is plotted in the `(x,v)` plane. However, the program is very easily modified to plot time versus position, that is, the motion is plotted in the `(t,x)` plane; see the instructions in the program.

   Two snapshots of GUIs created by the program are given on page 3 and page 4.

   Plot each of the following motions in both the `(x,v)` and `(t,x)` planes. You will need to change the function `__f` appropriately in your `One_d_motion` class for each of the motions.

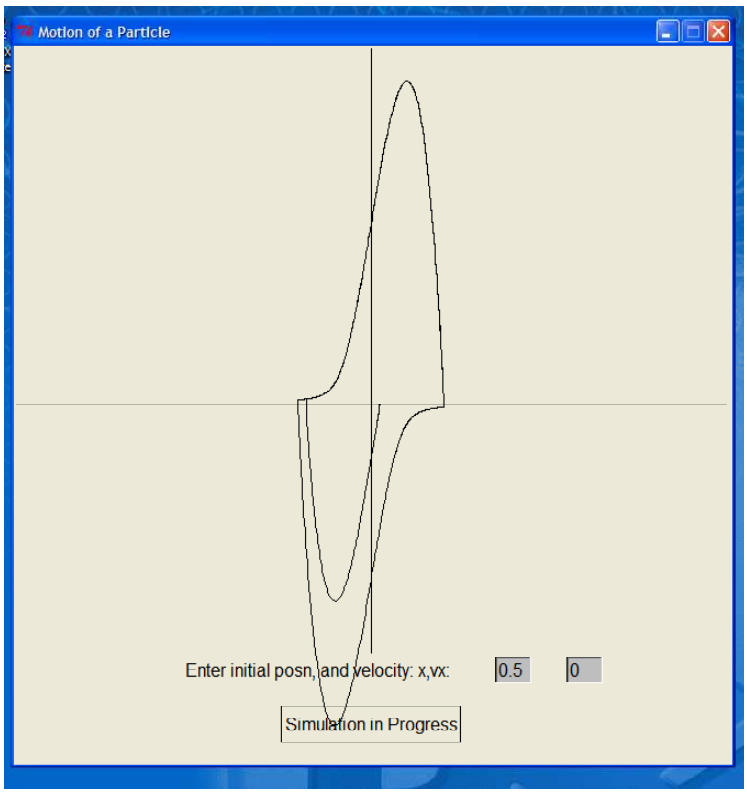
   (a) An undamped, unforced oscillator. Use $f(x, v, t) = -x$ and choose the initial (position,velocity) to be $(10, 0)$.

   (b) A damped, unforced oscillator. Use $f(x, v, t) = -x - 0.1v$ and choose the initial (position,velocity) to be $(10, 0)$.

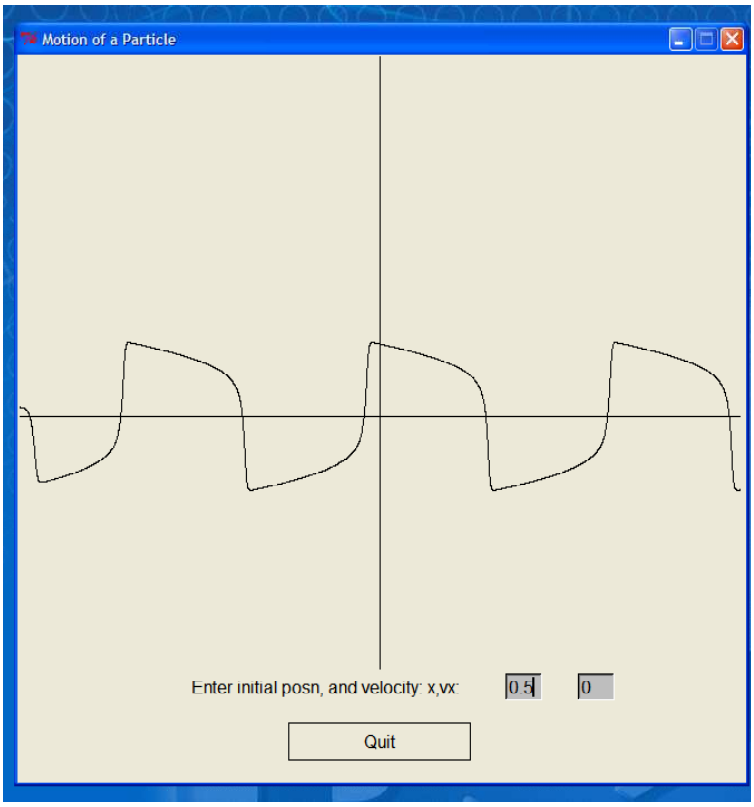
   (c) A forced oscillator at resonance. Use $f(x, v, t) = -x + \cos(t)$ and choose the initial (position,velocity) to be $(0.2, 0)$.

   (d) Duffing's equation. Use $f(x, v, t) = x - x^3 - 0.25v + 0.25\cos(t)$ and choose the initial (position,velocity) to be $(5, 0)$.

(e) A van der Pol oscillator. Use $f(x, v, t) = -x - 1.5(x^2 - 4.0)v$ and choose the initial (position,velocity) to be $(0.5, 0)$.



Snapshot of the GUI created by the plotter. Here $f(x) = -x - 1.5(x^2 - 4.0)v$, a van der Pol oscillator, and the plot is in the (x, v) plane.



The same van der Pol oscillator as plotted in the diagram above, but plotted now in the `(t,x)` plane.