Imperial College London

Department of AERONAUTICS

Computing and Numerical Methods 2
Coursework Part 2 Report

Aloysius Jiajun Yang
CID Number: 01937687

Date: 21 January 2023

Question 6

Part (a)

Table 1: Final positions and velocities of each case

Case	Body Index i	x_i	y_i	$\dot{x}_i$	$\dot{y}_i$
One Body	1	1	1	1	1
Deflect	1	0.26192	0.113273	0.30915	0.25463
	2	-1.61754	2.8327	-3.05681	2.45936
Orbit 1	1	4.07359E-5	0.0401739	-0.0091632	2.64926E-4
	2	0.961033	-0.0638629	0.951701	23.5386
Orbit 2	1	0	0	0	0
	2	-1.10045	1.05212	-13.8609	-9.46574
	3	1.10045	-1.05212	13.8609	9.46574

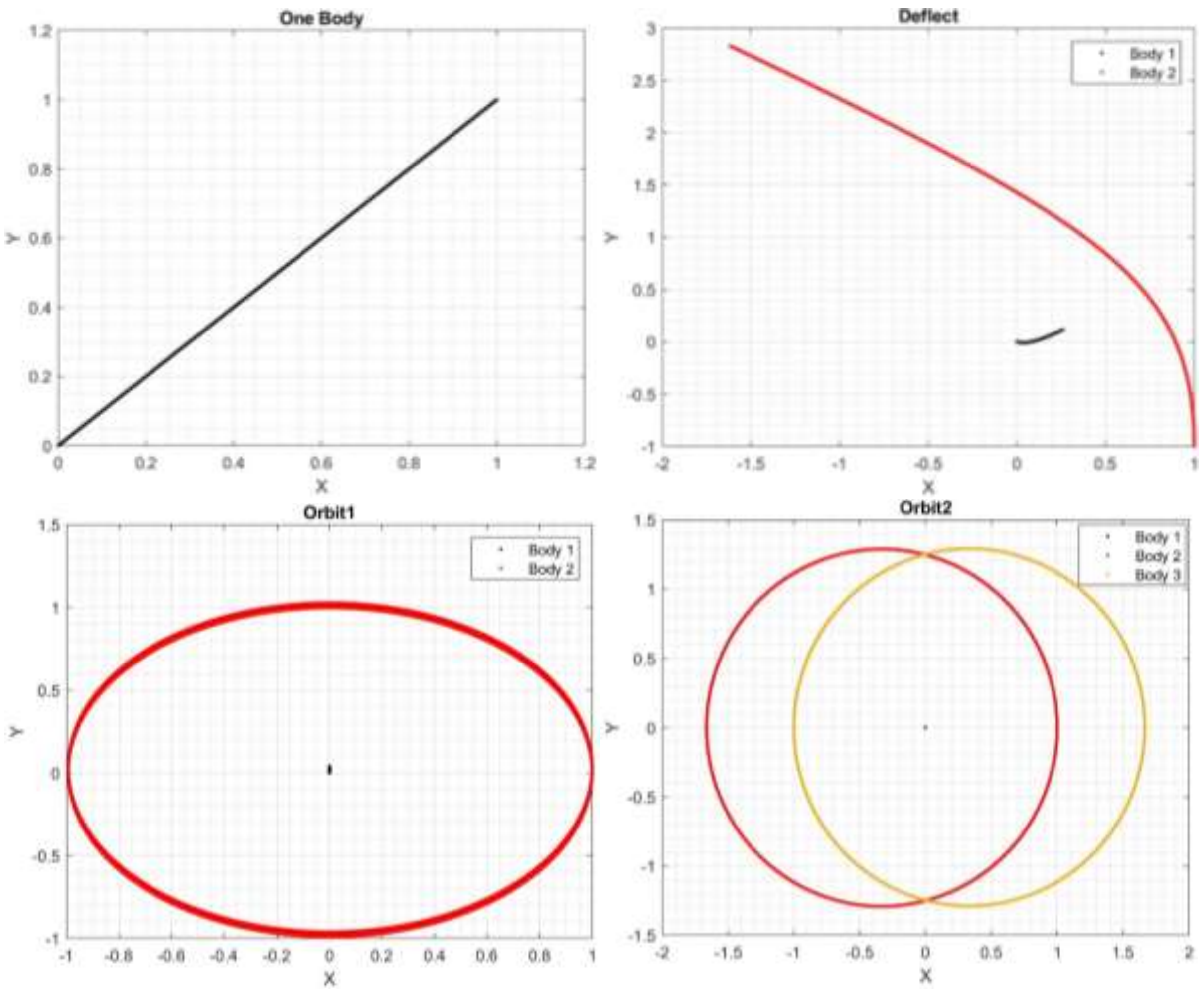
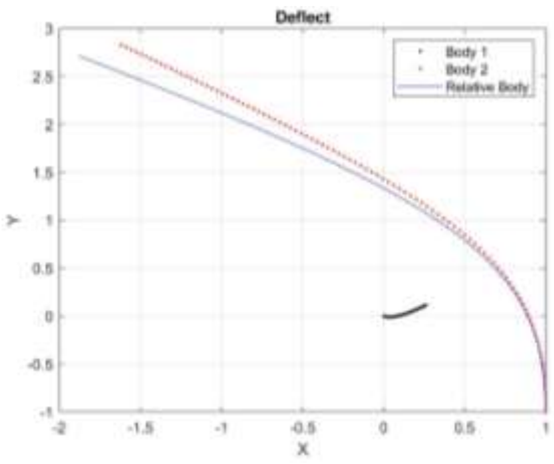


Figure 1: Trajectory plots of each case

In the MATLAB code, a relative body problem was used. As such, the one body case cannot be implemented here and only a trivial solution is produced. However for the Deflect and Orbit 1 cases, a valid comparison can be made. In the Deflect case, Body 2's position trajectory is rather similar except that the movement of the relative body in the MATLAB case deviates at a smaller slope (as seen in Figure 2).



In the Orbit 1 case, the trajectory of Body 2 in MATLAB mimiced that in C++. This therefore shows that if one mass is significantly larger than the other, the results will agree but if there is not much difference in magnitides, they will not.

Figure 2: Deviations of Body 2 in Deflect Case

Part (b)

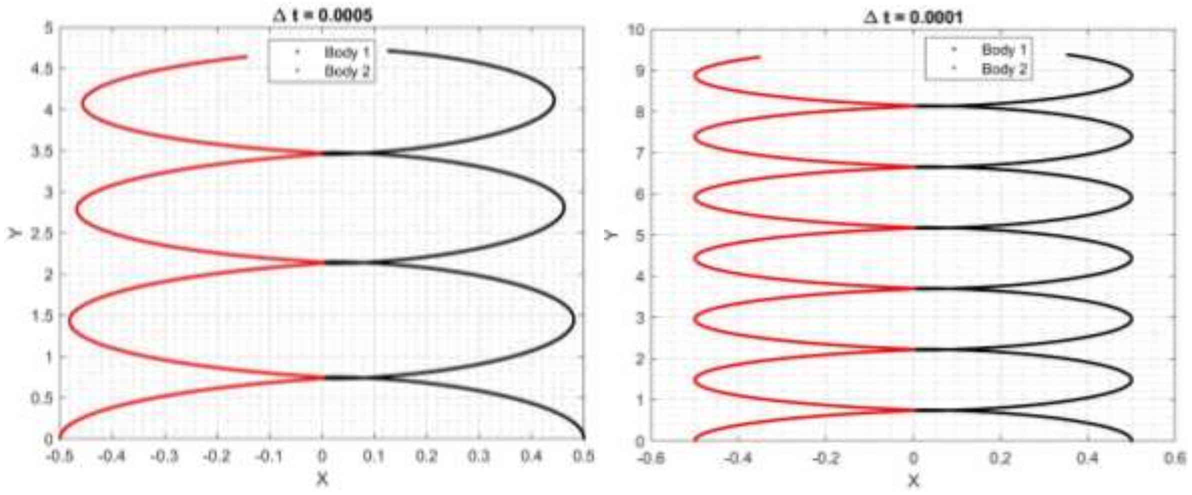


Figure 3: Trajectory plots for varying Δt

By testing various time steps, convergence only started to form at $\Delta t = 0.0005$. Once the Δt was dropped to 0.0001, there was no change in the orbit radius and number of loops. Hence $\Delta t = 0.0001$ provides a converged solution.

Part (c)

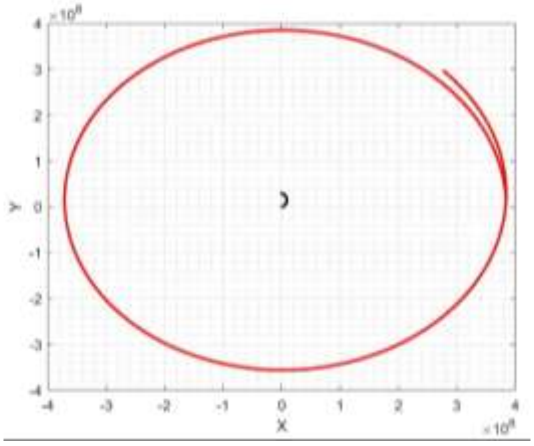


Figure 3: Earth Moon plot for 1 month

6.67E-11	2.63E+06	100	-	-
0	0	0	0	5.97E+24
3.84E+08	0	0	1.02E+03	7.35E+22

Table 2: Values inputted in parameters.txt file

Part (d)

The advantage of using C++ over an interpreted language would be its time efficiency. The speed at which the complied language of C++ executed upon compilation is faster than that of an interpreted language like MATLAB. This is because interpreted languages have to be parsed and interpreted then executed every time the code is ran, which requires more time. In addition, errors can also be spotted quicker as the code is compiled before being run, so errors can be picked up during the compilation process, as compared to interpreted languages whereby errors will only be found when the code is being run and time is wasted on running the steps before the error was caught. This therefore makes complied languages more time efficient, as they are not run on a step-by-step linear basis.

However, interpreted languages can be more advantageous as they are easier to implement and debug, and usually have pre-built standard libraries to solve problems like matrix calculations. For this particular problem, there were no standard libraries for matrix manipulation so it made the implementation of the code much harder and vectors within vectors had to be used, this would have been easier is C++ had inbuilt matrix calculus functions. Furthermore, C++ is statically typed so data types need to be declared initially.

Part (e)

Object-Oriented Programming in this code was used for abstraction, encapsulation using classes and making the solution neater and cleaner. OOP allows for functional abstraction. In functional abstraction, functions to transform the vector, calculate time derivative of a vector as well as to compute the 4th Order Runge-Kutta scheme can be abstracted from the main body of the programme. It also makes troubleshooting easier as it pinpoints where errors occur. It also allows us to build classes to act as a template for future uses. The code can be better upgraded and maintained to implement other solving methods of Runge-Kutta, Explicit and Implicit Euler and other relevant methods.

Instead of having to constantly call a function to perform a calculation, the functions were stored in classes so a simple call of the class would perform all the calculations from the functions. The body class contains information about its position, velocity and acceleration and use their justifications for it. This was implemented by encapsulating relevant functions in computing the 4th Order Runge-Kutta scheme together and reducing the need for many functional dependencies. Vectors were used extensively as data could be added when necessary, using vector pushback whenever an unknown number of values were inputted.