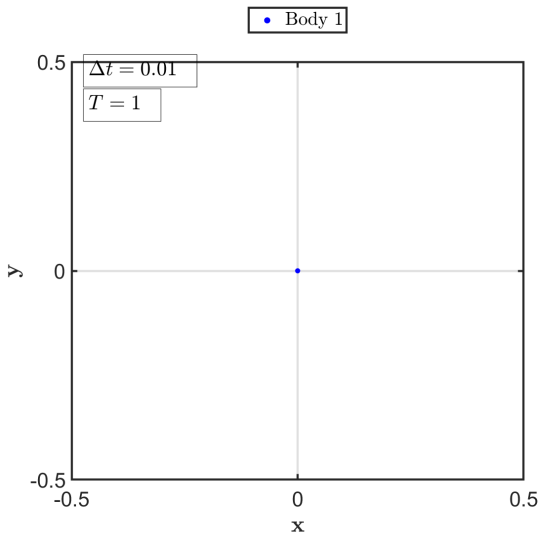
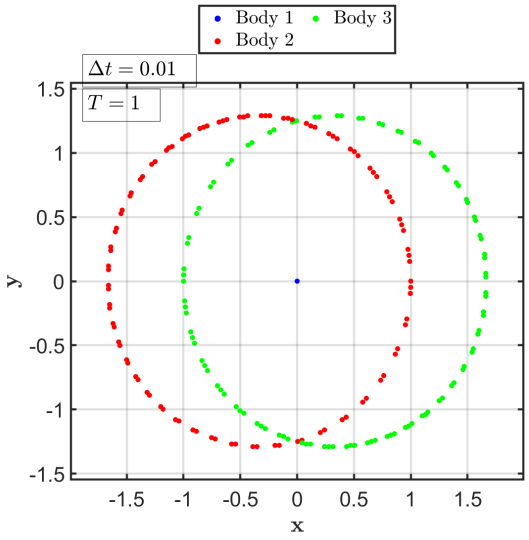
Question 6A

Figure 1: C++: Runge Kutta 4th Order for one (A) & three (B) body cases

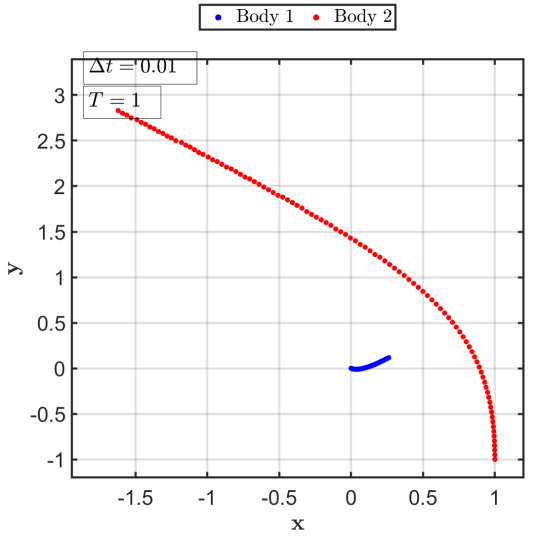


(A)

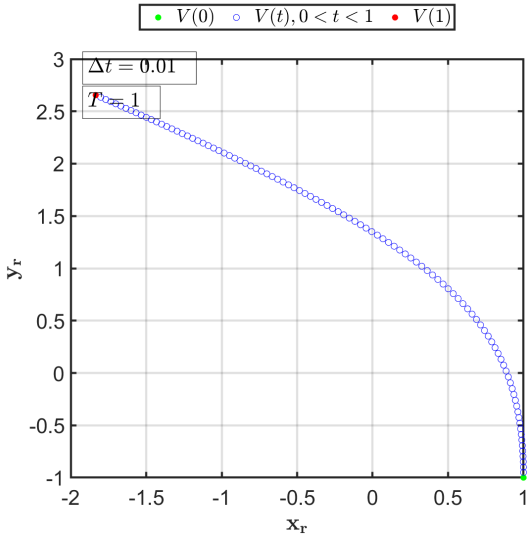


(B)

Figure 2: C++(A) & Matlab(B): Runge Kutta 4th Order for $m_1 = 10$kg

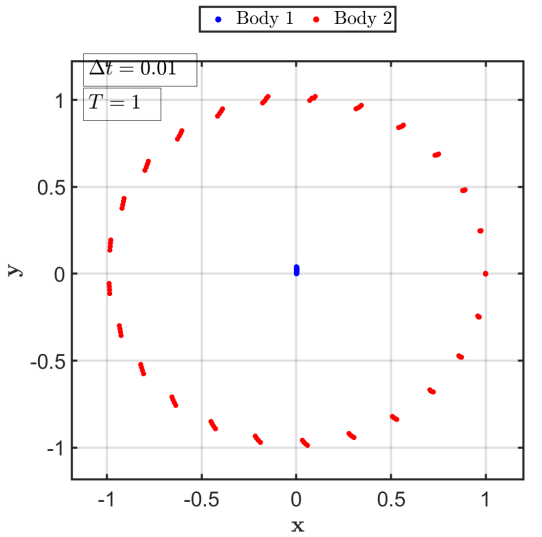


(A)

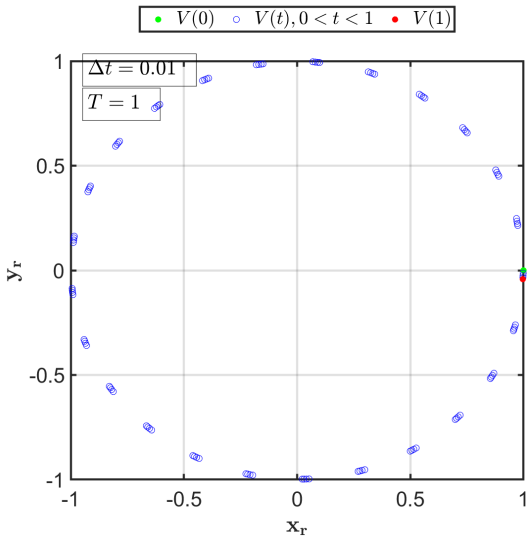


(B)

Figure 3: C++(A) & Matlab(B): Runge Kutta 4th Order for $m_1 = 625$kg



(A)



(B)

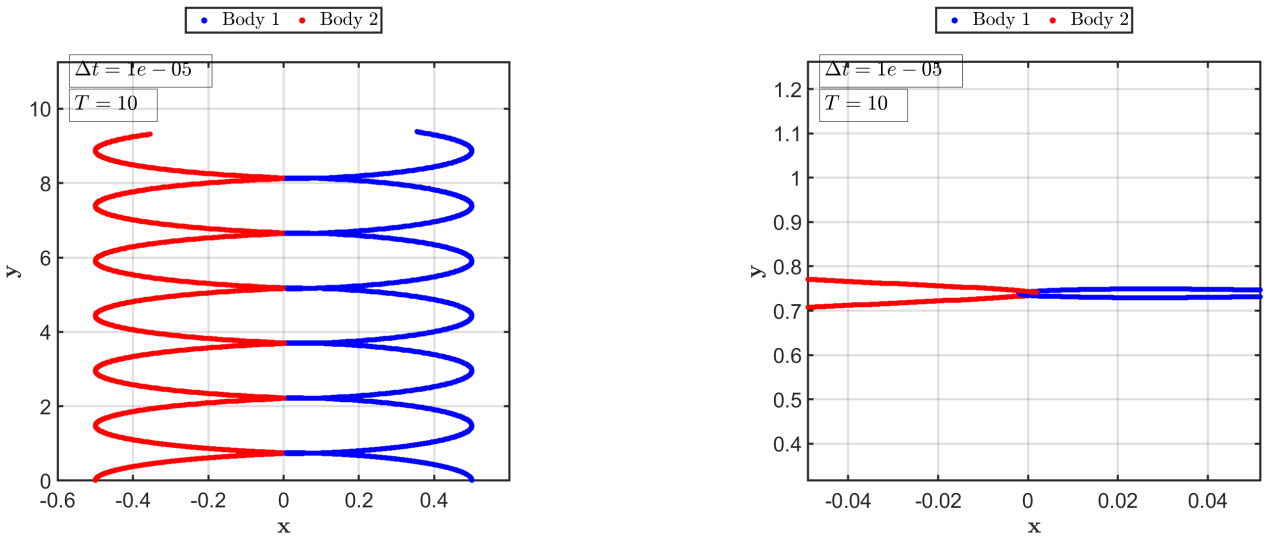
Table 1: Final positions and velocities of all bodies in requested cases

Body Index i	$x_i(T)$	$y_i(T)$	$\dot{x}_i(T)$	$\dot{y}_i(T)$
1	0.00e+00	0.00e+00	1.00e+00	1.00e+00
1	2.62e-01	1.17e-01	3.06e-01	2.55e-01
2	-1.62e+00	2.83e+00	-3.06e+00	2.45e+00
1	2.24e-06	4.00e-02	-1.49e-03	1.34e-05
2	9.99e-01	2.76e-03	9.32e-01	2.50e+01
1	0.00e+00	0.00e+00	0.00e+00	0.00e+00
2	-1.10e+00	1.05e+00	-1.38e+01	-9.46e+00
3	1.10e+00	-1.05e+00	1.38e+01	9.46e+00

In Figures 2 and 3, it is evident that the 4th Order Runge Kutta (RK4) scheme implemented into C++ aligned with the results found in Matlab. Whilst in the corresponding Matlab plots relative distances of the lighter body upon the larger one was plotted they clearly show the same motion.

Question 6B

Figure 4: C++: Runge Kutta 4th Order for Q6B



(A) (B)

In contrast to 3B, from the plots shown in Figure 4, the step size required for the convergence of this system was much lower and required much more computations.

$$\lambda = \pm \sqrt{\frac{G(m_1 + m_2)}{\sqrt{x_r^2 + y_r^2}^3}}\, i \tag{1}$$

In 3B the eigenvalues whilst made large by the values of the masses are made constant by the fact that the distances are constant. In contrast, the Figure 4B in which the relative distance between the two bodies gets smaller and even close to zero as they approach each other, leading to extremely large eigenvalues as given by (1). This is indicative of a rather stiff system of equations and therefore a much smaller step size is needed to bring stability in this system. The constant amplitude of orbits shown in Figures 4A and 4B indicate the solution has dynamically converged.

Question 6C

Table 2: Earth and Moon parameters.txt

6.67e-11	4.72e+06	1.00e+03		
0.00e+00	0.00e+00	0.00e+00	0.00e+00	5.97e+24
-3.84e+08	0.00e+00	0.00e+00	1.02e+03	7.34e+22

Figure 5: Earth and Moons orbits, produced from Runge /Kutta 4th Order (C++) with Table 2

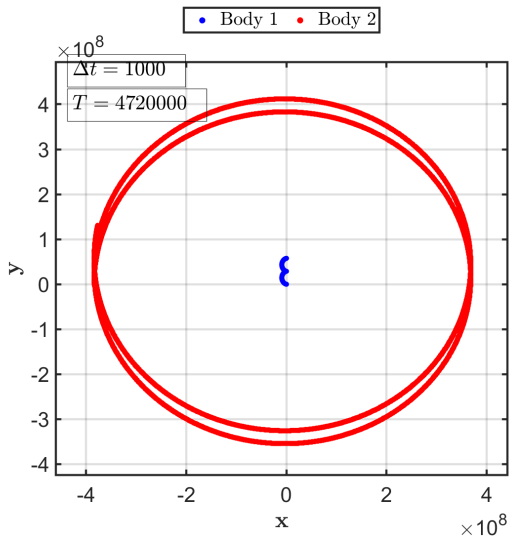
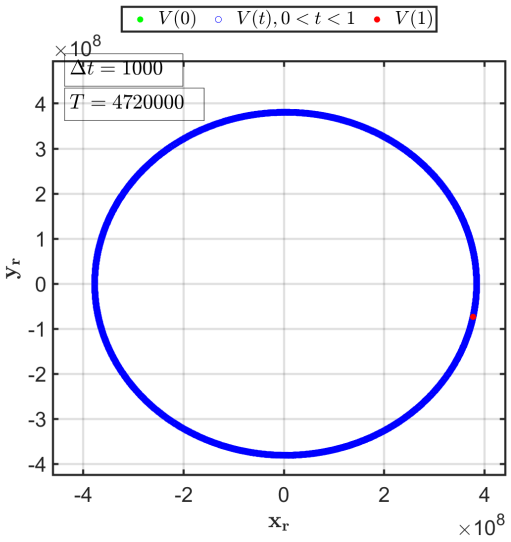


Figure 6: Moons Relative motion to Earth, produced from Runge Kutta 4th Order (C++) with Table 2



Question 6D

One of the main advantages with using a compiled language for this problem is faster execution time. A compiled language means that the code is translated into machine code before it is executed. This results in faster execution times compared to interpreted languages like Matlab. Another key feature is memory management. C++ allows for more control over memory allocation, which can be useful for solving problems that involve large amounts of data or need fine-tuning of system resources such as the 2nd order problem were solving. With C++ the source files are cross-platform compatible meaning that more collaboration between developers for simulation programs like these. C++ also has better performance for numerical computation as it has built-in support for mathematical operations and can work with raw data in ways that are not possible in interpreted languages.

The main disadvantage with C++ is the steep learning curve with writing and debugging the program itself as it is a more complex language inherently. Whilst access to higher complex tools like memory management is useful; in practice this leads to a much higher chance of causing a memory leak and segmentation fault as the memory management with some data types are all manual. In general, C++ code can also be generally less readable than interpreted languages as the notation in C++ must be robust. A big drawback in a use case like this, is the inability to easily plot and verify results against real world data.

Question 6E

In order to make my code robust, readable and most importantly reusable I took advantage of various Object-Oriented Programming (OOP) paradigms. Firstly with my use of the base class ”Body” and its private members, allowed me to implement a level of encapsulation with its setters and getters improved the integrity of the overall code. Furthermore in order to condense my code and ultimately improve readability I used the derived class, of base class ”Body”, ”Constants”. This allowed me to simplify my code and take advantage of what OOP has to offer developers, inheritance and polymorphism. Furthermore I implemented the overloaded operators in both classes for a more intuitive manipulation of objects and improved readability. In order to take advantage of the benefits of the Standard Template Library, through the use of STL container ”vectors”. With the ”vector” of objects i was able to iterate through various bodies with ease. Moreover the use of the ”vector” container meant that I did not have to worry about manually managing the memory whilst still taking full use of dynamic memory allocation. I also used various headers from the STL that allowed me to catch and handle errors, verify data input and print data out in a highly readable way. Overall, my code makes effective use of both the STL and object-oriented programming paradigms to improve the efficiency, maintainability and readability of the code.