Exercise 4 Report – Computational Physics

Caspar Fuller – 1310006

*Problem 1 – Basic Orbits*

The aim of this problem was to employ the fourth order Runge-Kutta method to simulate the orbit of a small body around a massive one, in this case the Earth was used. I shall investigate how the initial orbital velocity affects the eccentricity of orbits and how the energy within the orbit changes over time. The Runge-Kutta method is employed here to solve the equation of motion of the orbiting body seen in equation (1):

In the case modelled it was taken that *m<<M* and therefore negligible so can be ignored in the calculations. When the energy was modelled it was assumed to have a mass of 1kg which isn’t unreasonable for an orbiting probe.

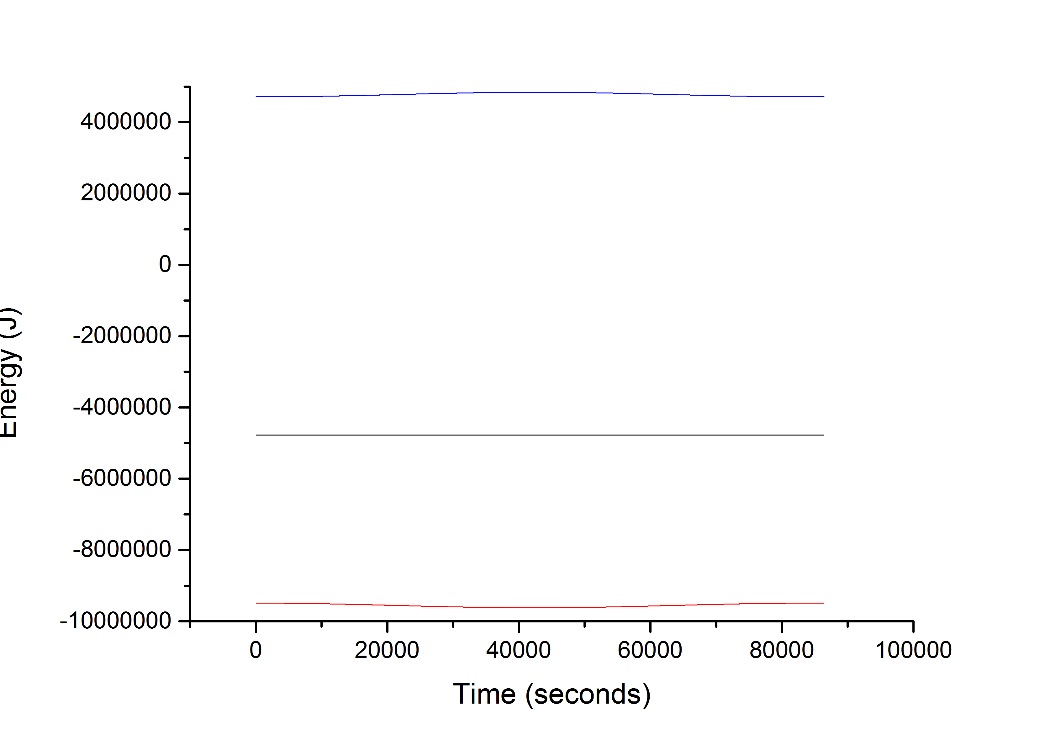
To simulate this order the program first asks for the initial conditions of the orbiting body, specifically the initial orbital velocity, radius of orbit, the angle defining the position of the body and the time for which the orbit shall be modelled for. Then using basic trigonometry values can be found that can be computed into the Runge-Kutta method. The program then iterates until the requested end time is met, each time defining values for the 16 k values and incrementing the values for t, x, y, Vx, Vy, KE, U and E. These are written to a file for later analysis. External functions were used in the program as half of the k-values required a complex function so it was preferable to call this function in each time, changing each variable when required.

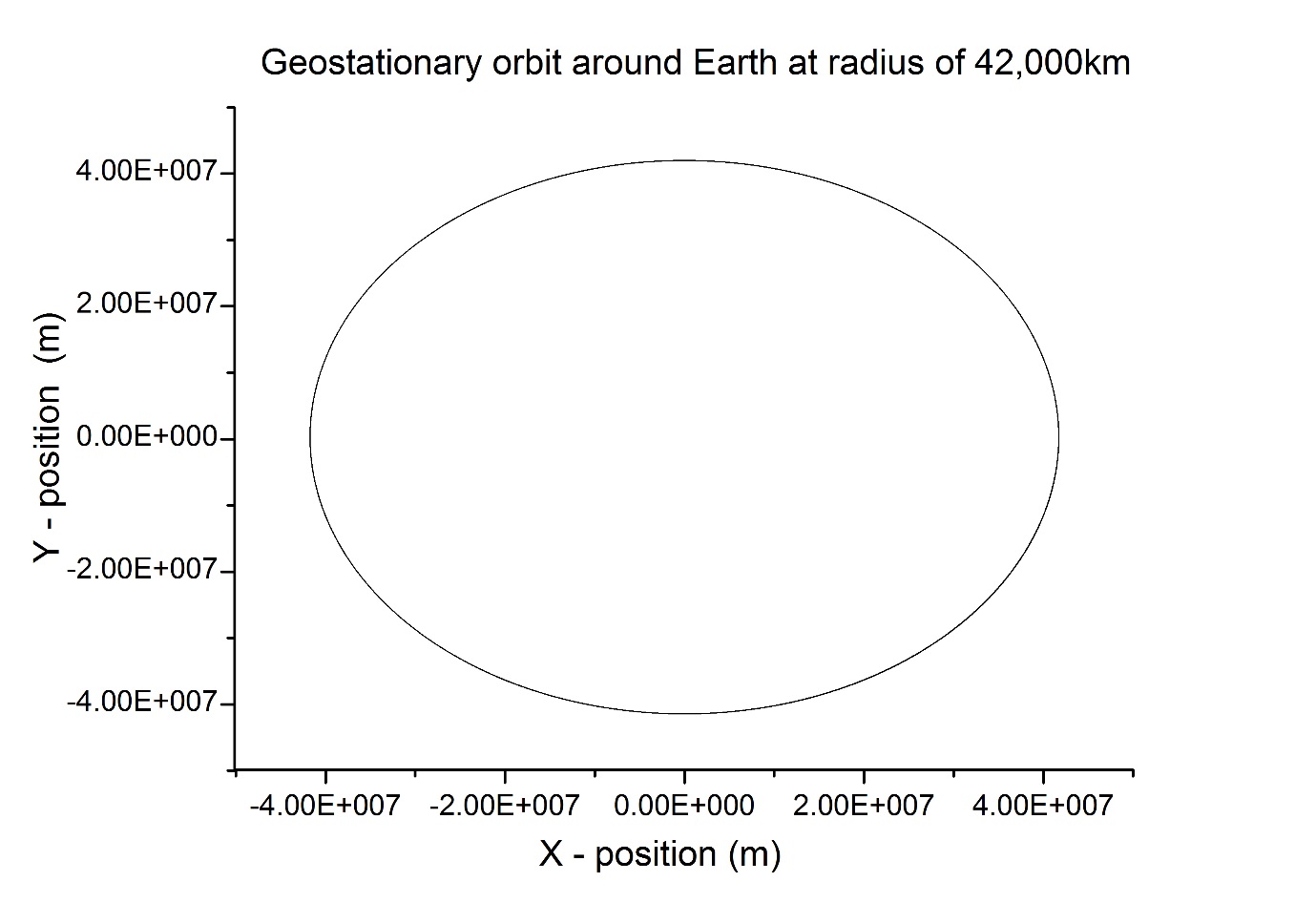
The step size for this program caused some issues at first. Initially a larger step size h=100 was used to speed up the run time of the program however the small errors in each iteration summated to a very inaccurate orbit model over long periods of time. Using a value of h=1 and a time step of one second led to longer run times, but much more accurate orbit models so the value of h was held constant at h=1 for the duration.

Finally a check was in place after each iteration to alert the user and stop the program if the radius of orbit was less than the radius of Earth. This would been the body had crashed into Earth and if the program was allowed to continue iterating the method was seen to break down as the body reached the centre of Earth. The values for G\*Me where Me is the mass of the further were explicitly defined as when tested it was found that the computer often made errors when multiplying very large values with very small values making the model inaccurate.

Firstly a circular orbit was obtained from the program. The initial conditions for this were as followed;

* Radius of orbit = 42,000km (corresponding to a geostationary orbit)
* Initial orbital velocity = 3070km/s
* Time to track orbit for = 1 day (this is converted to seconds and corresponds to one complete orbit for a geostationary satellite.)
* Initial angular position of orbit (anti-clockwise from the positive x-axis) = 90 degrees.

Figure (1) shows the results of this orbit. The orbit was found to agree with the theoretical values for a geostationary orbit and when run for an extended period it proved to be a very stable orbit. Figure (2) shows the energy diagram for the orbit and showing that the overall energy, black line, remains constant and negative which is as expected meaning the satellite will stay in orbit.



*Figure 1: A geostationary circular orbit simulation. Earth is centred on x=0 and y=0.*

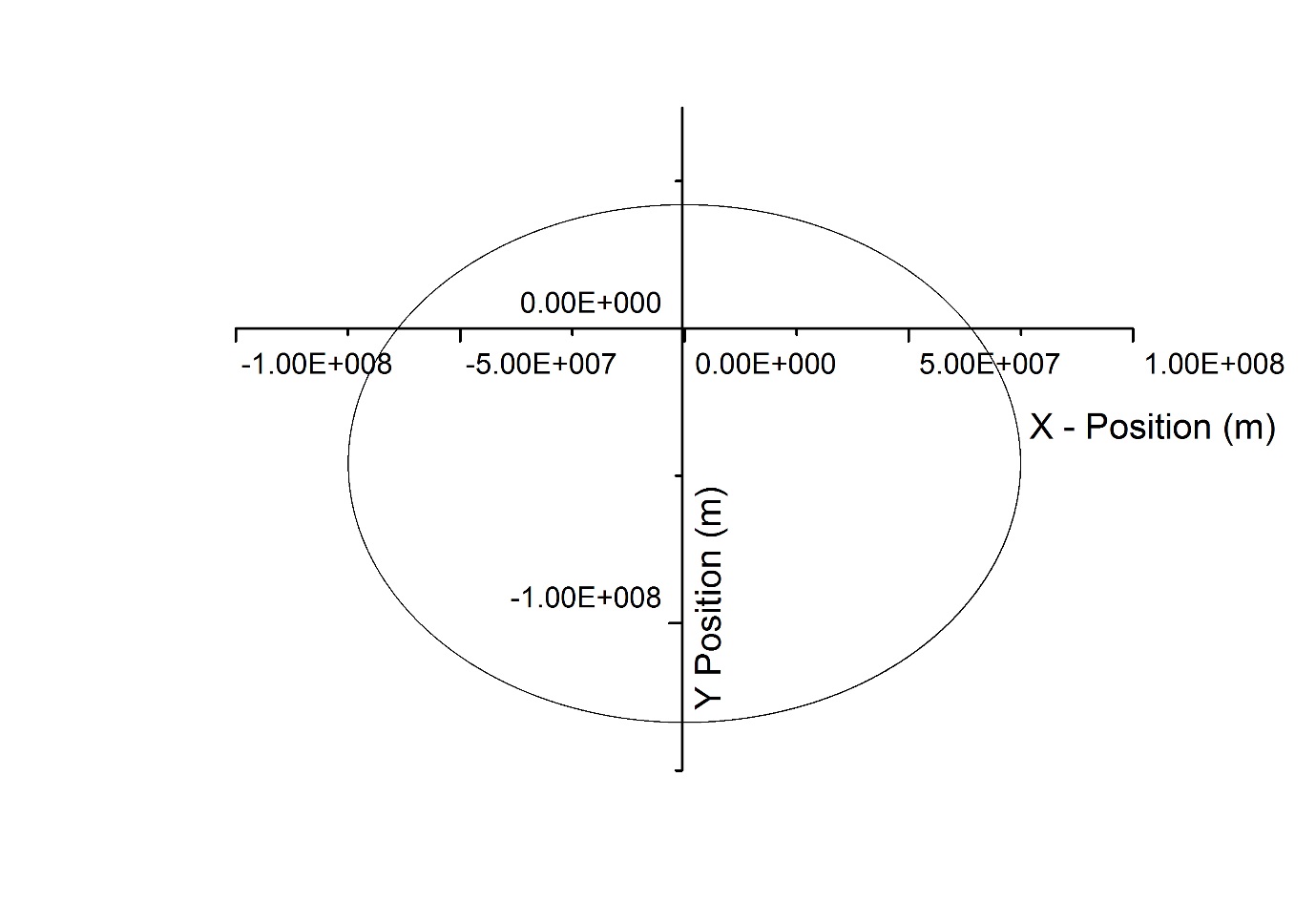
Blue Line – Kinetic Energy

Black Line – Overall Energy

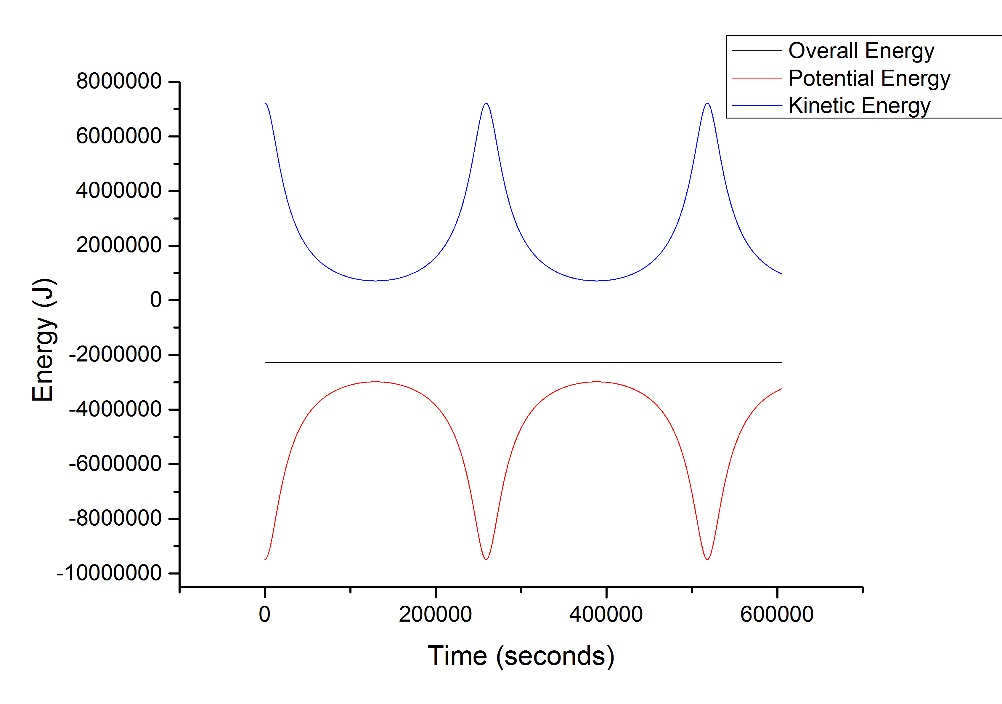
Red Line – Potential Energy

*Figure 2: Plot of variation in energies with time for the circular orbit.*

Next an elliptical orbit was found. The only initial conditions to be changed was the initial orbital velocity which was set as 3500 m/s and the time for the orbit to be tracked for. This was set as 7 days to investigate the stability of the orbit. Figure (3) shows the resulting orbit, proving the program functions for orbits of varying eccentricity. The energy is once again plotted and shown in figure (4) as the time period is greater here we can see a pattern of fluctuations within the orbits. As it is an elliptical orbit there is more variation in the kinetic and potential energies with a maximum in kinetic energy corresponding to the closest approach to the earth the satellite makes. This also corresponds directly with the minimum for the potential energy as expected, resulting in a constant negative overall energy once again. This overall energy is greater than the previous circular orbital energy, corresponding well to orbital theory that elliptical orbits have a greater energy than circular orbits. Also no variation in the orbit can be seen, even though the satellite completes multiple orbits proving the stability of the orbit.



*Figure 3:* *A simulated elliptical orbit around Earth. Earth is centred on x=0 and y=0.*



*Figure 4: Plot of variation in energies with time for the elliptical orbit.*

Overall I found the code worked sufficiently well to complete the task at hand. An improvement for the code would be to find a method to solve the eccentricity equation within the program, allowing for the user to enter a value for the eccentricity of the orbit they desire and then for the program to calculate what initial velocities can do this. This method was approached but it was found that the computer could not accurately compute the results leading to inaccurate orbits. However I’m sure with more careful calculations this could be completed, simplifying the initial conditions greatly.

*Problem 2 – Moon Shot*

The aim of this problem was to model a real-world situation of launching a probe from a constant orbit around Earth so that it passes around the moon in order to take photographs and then passes by Earth again transmitting the photos back to Earth. Once again the 4th order Runge-Kutta method shall be employed however in this situation the full boundary conditions of the problem are not known so there is a large element of trial and error within the building of this program.

As this program used the Runge-Kutta method again the code for problem 1 could be adapted to solve this problem. The solution for the equation of motion changes in some respects as shown in equation (2) which shows the new solution for finding , accounting for the mass of the moon.

Where Mm is the mass of the moon and Rm is the x-component of the distance from the moon the probe is. The value Rm can be used here due to the symmetry of the system applied in the program. The Earth was centred on the origin and the moon was fixed therefore at a distance of 384,400,000m along the x-axis meaning that the y-component does not need to be changed when solving the equation of motion.

As the aim of this program was to complete a slingshot around the moon, which proved to be a very precise orbit I found it inappropriate to allow the user to input values, as finding the correct initial conditions is a long process. It was found that the correct initial conditions varied dependent on a variety of seemingly arbitrary conditions, such as the computer used to run the program. The initial conditions were therefore held constant to make a successful slingshot occur. This included setting the angle for the probes impulse to take places which was found at 180 degrees from the positive x-axis as seen from the Hohmann transfer orbit equations. The initial orbital radius was fixed at 7,000km from the centre of Earth and the initial impulse of the probe was found via methods described below to be 3011.97 m/s. The program then calculates the sum of the orbital velocity of the probe needed to keep it at a constant 7000km orbit and the impulse given to the probe giving a final initial velocity.

The program iterates over a million time steps, found to be ample time for the probe to reach the moon and come back, with a constant step size of h=1 giving accurate results which are written to a file. The iterations are done over the same 16 k values and the same increments for x, y, Vx, and Vy as before just with the external function to calculate and changed to account for the mass of the moon. As before values for G\*Me and G\*Mm were explicitly expressed to reduce error and checks were made if the use of an if statement to notify the user if the probe has crashed into the Earth or the moon.

One change from the previous code is that a value for how far the probe is from the moon, r1, is also written to the file. This is calculated using a simple Pythagoras calculation and allows for an easy method of finding the point of closest approach between the moon and probe. By plotting time against r1, shown in figure (5), it is possible to determine at what time the probe gets closest to the moon, then by searching through the data the exact x and y positions of the probe can be seen for this point. As the moon is at the co-ordinates x=384,400,000 and y=0 it is possible to compute another simple Pythagoras calculation to find the true closest approach of the probe and the moon.

*Figure (5): Plot of distance of probe from the moon against time*

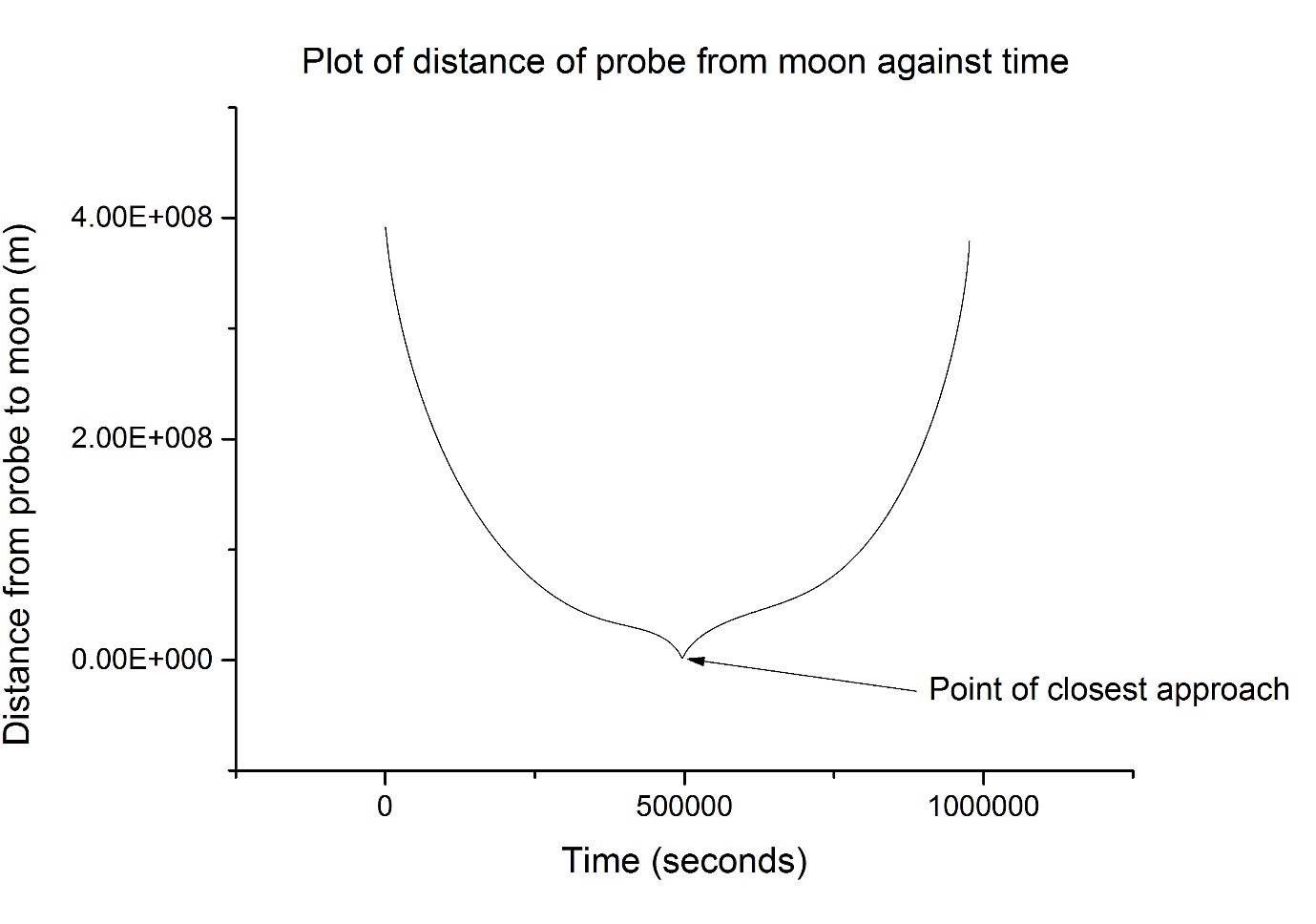
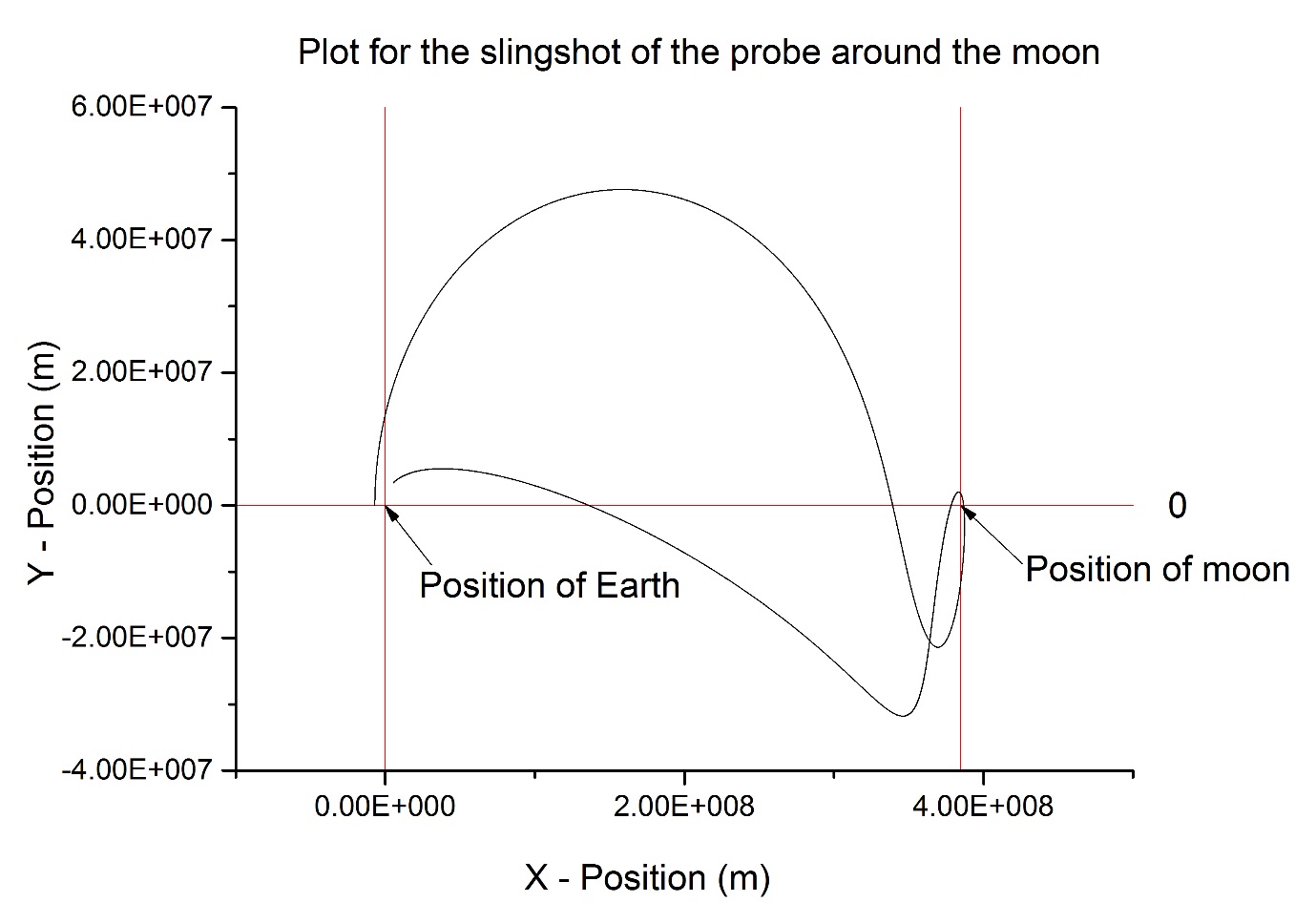
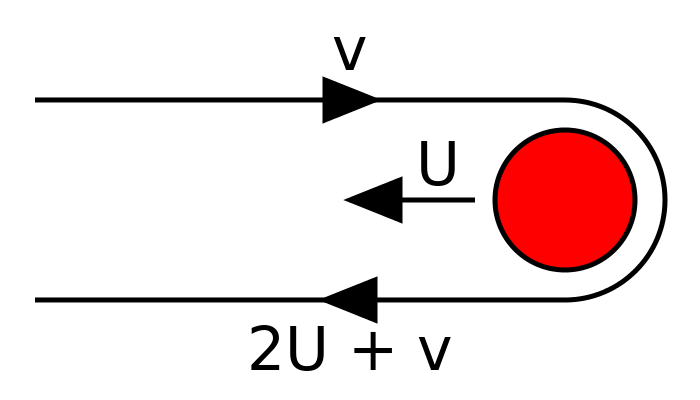


Figure (6) shows the complete orbit of the probe. The closest approach was seen to be at 24km which is a very good result as it was found that even the smallest variations in velocity affected the orbit greatly. An initial impulse of 3011.96 m/s for example caused a crash into the moon, and 3011.98m/s caused a closest approach of over 500 km and so outside the required accuracy. Unfortunately the probe crashes into Earth on its return however there should be enough time on its re-entry to successfully transmit the data back to Earth. It could also be possible to equip the probe with a parachute and beacon allowing it to be retrieved when it crashes into Earth. Once again varying step size was tested, and it was found that for values above h=5 the orbit again becomes riddled with error leading to an inaccurate orbit plot, therefore once again h was fixed as h=1 for the program.



*Figure (6): Plot for the slingshot of the probe around the moon*

The moon shot calculated in this program was a very good shot, meeting the criteria of the problem well. However when analysed the true accuracy of the orbit can come into question. To improve the code it would be useful to put in calculation’s for the perturbation of the orbit caused from other massive planets, the sun and even variations in mass concentrations within the moon itself. This may cause significant changes in the orbit of the probe making my model inaccurate. Also the moon was assumed to be in a fixed position, which is another approximation which could lead to errors, as if the moon was modelled to be orbiting the Earth then there would be significant changes in the velocity of the probe as it slingshots around the moon as shown in figure (7). These improvements would lead to a more accurate model of orbit. When finding the closest approach it was important to take into account the radius of the moon which is approximately 1700km as without this finding a close approach would mean the satellite had crashed into the moon. The problem states that the moon shot should be independent of step size, *h,* however as discussed previously this was found to be wrong as a change in step size causes large fluctuations within the orbit. Finally it was found that there were multiple initial impulses that caused a successful slingshot around the moon, therefore the addition of a for loop which iterated through a range of initial impulses could be employed to investigate which one of these gave the best results

*Figure (7) [1]: Change in velocity when performing a slingshot around a moving body.*

References

[1] - http://en.wikipedia.org/wiki/Gravity\_assist#/media/File:Gravitational\_slingshot.svg