University of Nottingham
School of Mathematical Sciences

MATH4063/G14SCC SCIENTIFIC COMPUTING AND C++

Deadline: 7th January 2022, 5:00pm (GMT) **Coursework 2 – Solution Template**

Your solutions to the assessed coursework may be submitted using this template. Please cut and paste the output from your codes into the correct parts of this file and include your plots and responses to the questions where suggested. Once this template has been completed, you must then create a pdf file for submission. Under Windows or Mac you can use Texmaker + a LaTeX compiler; from the Windows Virtual Desktop this may be accessed as follows:

Start > UoN Application > (UoN) Texmaker 5

Open this file under File*; to build the pdf file, click the arrow next to* Quick Build*; this will then generate the file* coursework2 submission.pdf*.*

You may use an alternative document processing system, such as Word, to produce a pdf file containing your results, plots and answers. However, if you do, you must format your answers in the same way as suggested below.

A single zip or tar file containing the file coursework2 submission.pdf *and all the files in the requested folders in the checklists below should be submitted on Moodle. Note that all parameters and values should be set within your codes: do NOT use inputs such as those obtained with* std::cin *or from the command line.*

WEICHENG YU

20124945

* Avoid using the folder in the default filenames, e:\\output.dat does not work on all systems.
* Use assert in Set functions and then use the Set functions in the constructor.
* Your while loop stopping criterion will be unreliable due to rounding error.

Q1 a 13/15
b 5/5
c 14/15
d 5/5

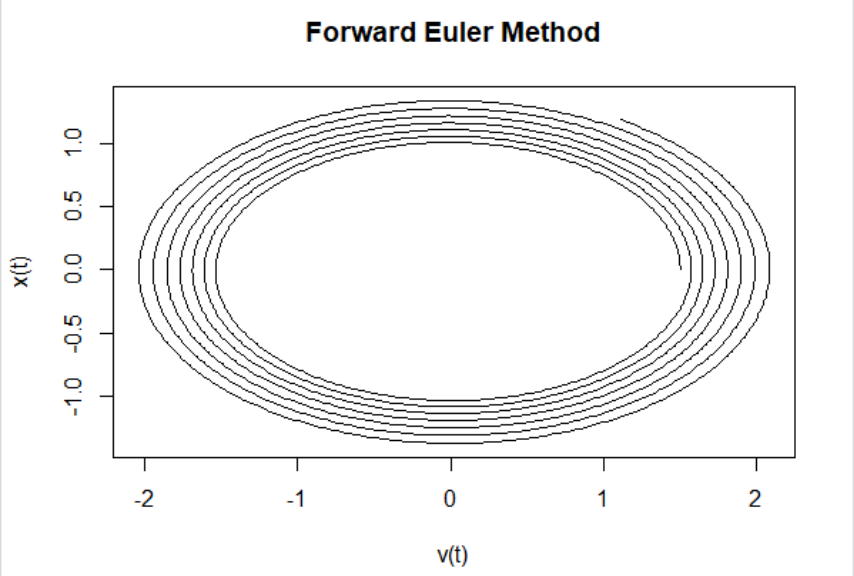
File checklist for folder Q1:

- AbstractODESolver.cpp, AbstractODESolver.hpp
- Driver.cpp
- ForwardEulerSolver.cpp, ForwardEulerSolver.hpp
- ODEInterface.cpp, ODEInterface.hpp
- OscillatorODE.cpp, OscillatorODE.hpp
- For any additional files, provide a README.txt

* Enter your output here (max 1 page, display only selected output if necessary).

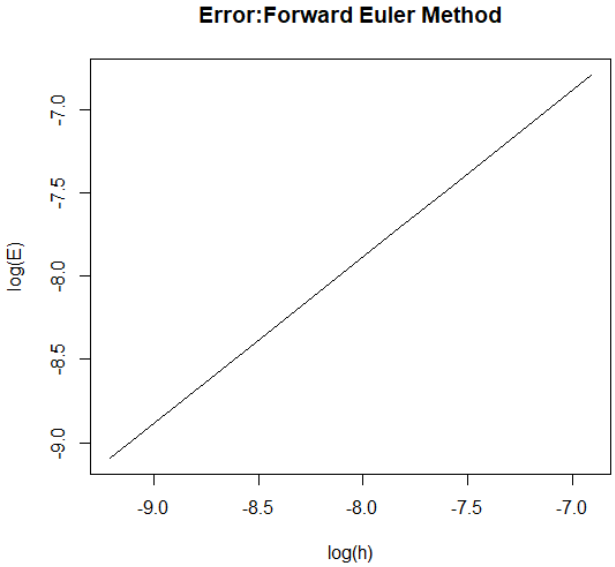
%%%%%%%%%%%%%%%%%% Output of Driver.cpp %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
In (d)ii
h={0.0001,0.0002,0.00025,0.0004,0.0005,0.0010}
E(h)={0.000112236, 0.000224508, 0.000280658, 0.00044916, 0.000561539, 0.00112273}
In my output:
0.0001 0.000112236
0.0002 0.000224508
0.00025 0.000280658
0.0004 0.00044916
0.0005 0.000561539
0.001 0.00112273
%%%

* Include your plots and comments here.
(d)i



Comment: The plot is quite similar to an ellipse. Actually the plot that x(t) against v(t) should be an ellipse. Because the exact solution is $x(t) = \sin(at)$ so that $v(t) = x(t) = a * \cos(at)$. Hence $\frac{v(t)^2}{a^2} + x(t)^2 = \cos^2(at) + sin^2(at) = 1$ which is an ellipse. The plot above is similar to an ellipse but not completely an ellipse which indicates that the there exist error by using forward Euler method.

(d)ii



I use h={0.0001,0.0002,0.00025,0.0004,0.0005,0.0010} and E(h)={0.000112236, 0.000224508, 0.000280658, 0.00044916, 0.000561539, 0.00112273} to plot. I choose the set of h because I need to make sure the number of time steps Nt=(T-t0)/h is an integer and I need to make sure that h is sufficiently small value to get sensible approximation.

Here I can see that the relationship between log(E) and log(h) are linear so that log(E)=k*log(h)+b which means that E=A*(h^k). According to the plots above, I could calculate k=1 approximately. Hence, E=A*(h^1) approximately so that E(h)=O(h^1) here.

Q 2a 5/5
b 5/5
c 10/10

File checklist for folder Q2:

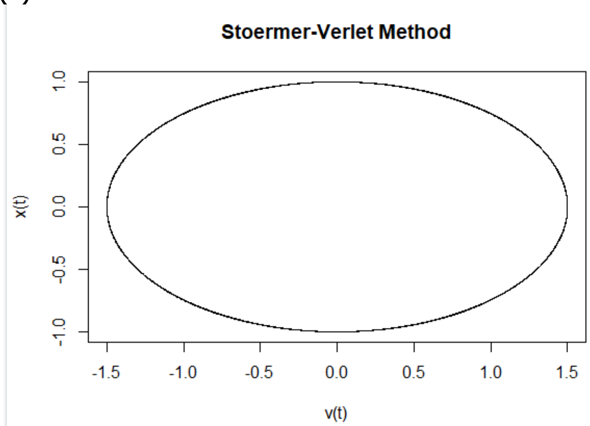
- AbstractODESolver.cpp, AbstractODESolver.hpp
- Driver.cpp
- ODEInterface.cpp, ODEInterface.hpp
- OscillatorODE.cpp, OscillatorODE.hpp
- StoermerVerletSolver.cpp, StoermerVerletSolver.hpp
- SymplecticEulerSolver.cpp, SymplecticEulerSolver.hpp
- For any additional files, provide a README.txt

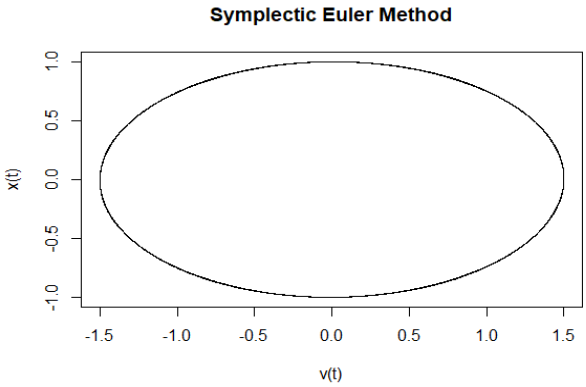
* Enter your output here (display only selected output if necessary).

%%%%%%%%%%%%%%%%%% Output of Driver.cpp %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
In (c)ii
h= {0.0001,0.0002,0.00025,0.0004,0.0005,0.0010}
E(h)= {3.03908e-09, 1.21562e-08, 1.89941e-08, 4.86248e-08, 7.59763e-08, 3.03905e-07} by symplectic method
E(h)= {3.03908e-09, 1.21562e-08, 1.89941e-08, 4.86248e-08, 7.59763e-08, 3.03905e-07} by Stoermer-Verlet method

In my output (the same for these two methods):
0.0001 3.03908e-09
0.0002 1.21562e-08
0.00025 1.89941e-08
0.0004 4.86248e-08
0.0005 7.59763e-08
0.001 3.03905e-07
%%

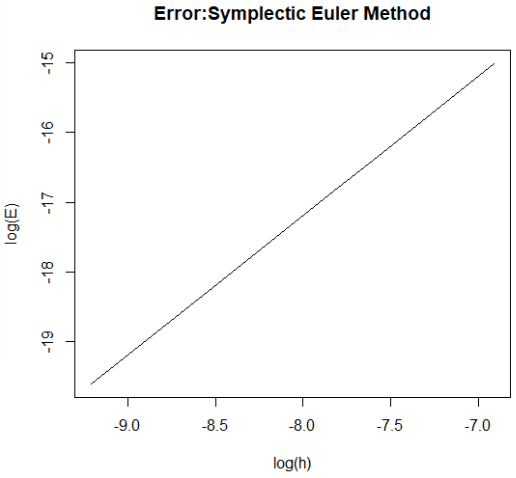
* Include your plots and comments here.
(c)i

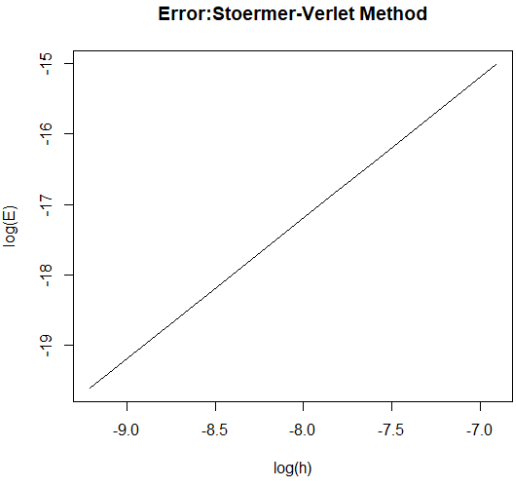




The two plots here are quite similar. As I mentioned in question 1, the exact plot of x(t)-v(t) should be an ellipse so that the more elliptic the image is, the more accurate the method is. Compared with forward Euler method, the plot produced by Stoermer-Verlet method and symplectic Euler method are much more elliptic so that these two methods are better. However it is quite hard to tell which of plots of Stoermer-Verlet and symplectic Euler is more elliptic. By my observation in R(which I used to plot), the plot of Stoermer-Verlet is a little bit more elliptic.

(c)ii





I use h= {0.0001,0.0002,0.00025,0.0004,0.0005,0.0010} and E(h)= {3.03908e-09, 1.21562e-08, 1.89941e-08, 4.86248e-08, 7.59763e-08, 3.03905e-07}to plot. Because in my output the x(t) produced by these two methods are the same, hence the error of these two methods are the same. Here I can see that the relationship between log(E) and log(h) are linear so that log(E)=k*log(h)+b which means that E=A*(h^k). According to the plots above, I could calculate k=2 approximately. Hence, E=A*(h^2) approximately so that E(h)=O(h^2) which is order 2 here(in both methods).Thus compared with forward Euler method which is order 1, these two methods are more accurate. Although x(t) and E(h) are the same in my result between these two methods, I can compare the v(t)

at T=30 (final time) produced by these two methods. Because at time T, the code has iterated for many times and can show the accuracy strongly. At time T, v(t)=0.787444 by Stoermer-Verlet methods and v(t)=0.797020 by symplectic Euler method. The exact solution v(t)=a*cos(a*t)=1.5*cos(1.5*30)=0.78798298 so that the v(t) produced by Stoermer-Verlet methods is more accurate hence I consider Stoermer-Verlet method as the best methods among these three methods. The second is symplectic Euler method and the worst is forward Euler method.

Q 3a 5/5
b 4/5
c 7/10

File checklist for folder Q3**:**

- AbstractODESolver.cpp, AbstractODESolver.hpp
- Driver.cpp
- ForwardEulerSolver.cpp, ForwardEulerSolver.hpp
- ODEInterface.cpp, ODEInterface.hpp
- OrbitODE.cpp, OrbitODE.hpp
- StoermerVerletSolver.cpp, StoermerVerletSolver.hpp
- SymplecticEulerSolver.cpp, SymplecticEulerSolver.hpp
- Vector.cpp, Vector.hpp
- For any additional files, provide a README.txt

* Enter your output here (display only selected output if necessary).

%%%%%%%%%%%%%%%%%% Output of Driver.cpp %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
In (c)i
After one period, the state approximated by these three methods are: forward Euler method (3.87162e+08, -1.42709e+07,0), symplectic Euler method (3.844e+08,-675.399,0), Stoermer-Verlet method (3.844e+08,-317.835,0)
In my save output:
forward Euler: 2.37193e+06 3.87162e+08 -1.42709e+07 0 37.3446 1013.58 0
symplectic Euler: 2.37193e+06 3.844e+08 -675.399 0 0.00178912 1018.27 0
Stoermer-Verlet: 2.37193e+06 3.844e+08 -317.835 0 0.000841937 1018.27 0
I save the data in this form in order to be easily read by RStudio so that help me plot. ✓

In (c)ii Close, but not precise.
Collision happens after 6981 minutes and at this time x=(6.82578e+06,0,0)
%%%

* Include your plots and comments here.

(b)
I choose to define the DetectCollision in the class OrbitODE(I need to put a virtual method in ODEInterface) because in DetectCollision I need to use the value of Xp and the radius of Moon and Earth which is the private parameter that I need to put as a part of specialized constructor of OrbitODE. It is better in the solver.

(c)i
After one period, the state approximated by these three methods are: forward Euler method (3.87162e+08, -1.42709e+07,0), symplectic Euler method (3.844e+08,-675.399,0), Stoermer-Verlet method (3.844e+08,-317.835,0).
My order of these three methods from the best to the worst is: Stoermer-Verlet method, symplectic Euler method, forward Euler method. Reason: According to the question, the orbit of the moon should be a circle. Hence after one period, the best method is to produce the closest state to the initial state (3.844e+8,0,0). Here the state produced by Stoermer-Verlet method is the most closed to the initial state. The second is symplectic Euler method and the last is forward Euler method. The order is the same as I made in question 2 hence the Stoermer-Verlet method is the best to make the approximation. ✓

(c)ii

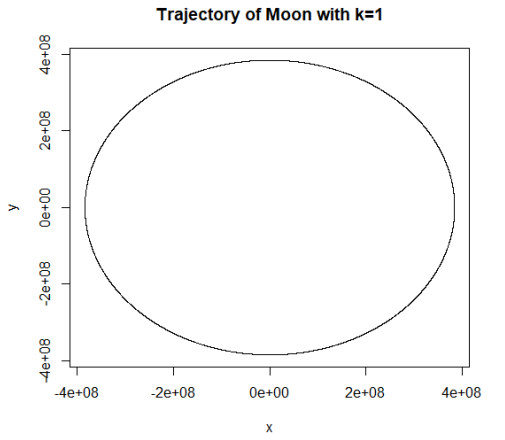
How do you choose k?

In the situation of this question, the initial velocity of the moon is (0,0,0) so that the trajectory of moon should be a straight line and free fall to earth. In my result produced by Stoermer-Verlet method, the state of moon is in a straight line and is (9.15769e+6,0,0) before the collision and (6.82578e+6,0,0) at the collision. Hence the distance r between moon and earth is 9.15769e+6 before the collision and 6.82578e+6 at the collision, which satisfies that r>rMoon+rEarth(8.115e+6) before the collision and r<=rMoon+rEarth after the collision. Thus I think the accuracy of my solution is not bad.

If I assume that the bodies are point masses, then the collision will not happen. This is because that if they have zero radius, the collision will happen if and only if the state of the moving body is on the state of the fixed body i.e. x=xp. However it is impossible to reach it because in this question the right-hand side function is $x(t) = \frac{\ddot{G}m_p(x_p - x)}{|x_p - x|^3}$ and if x=xp then the denominator of function $|x_p - x|$ becomes to 0 which is impossible.

What does your code do?

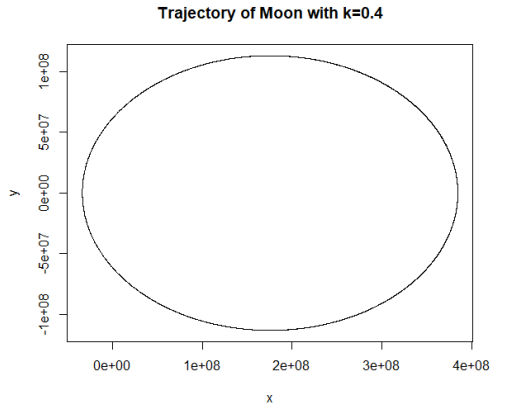
(c)iii

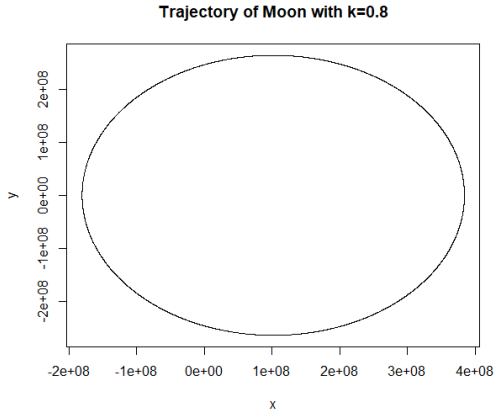


✓

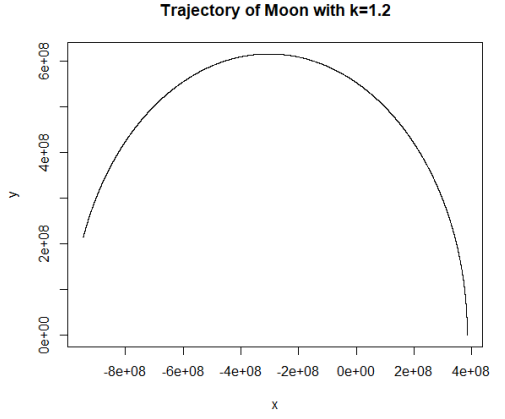
This is the satandard trajectory of moon produced in the (c)i with the initial velocity (0,|v0|,0).

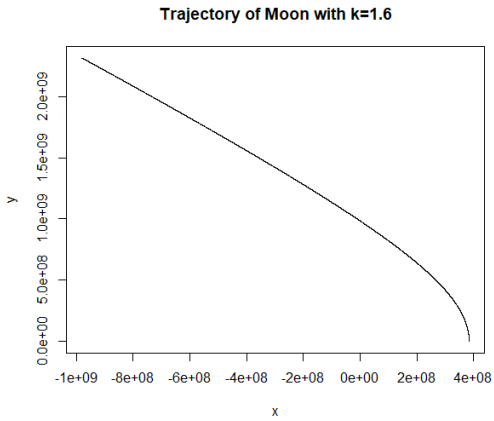
In this question I try different k={0.4,0.8,1.2,1.6} of (0,k*|v0|,0). The reason I choose these four values is that I want two choose 2 values that smaller than 1 and 2 values that bigger than 1. Both k>1 and k<1 will cause different abnormal situations so that I want to compare these different situations. Here are the plots:





✓





We can see as k=0.4 and k=0.8 which are k<1 the trajectories are ellipses. As k=0.4, the scalar of the initial velocity is very small and the earth (0,0,0) is quite close to the trajectory of the moon. The moon tends to strike the earth. As k=0.8, the shape of the trajectory is closed to a circle. As k=1.2 and k=1.6, the k tends to bigger. Then moon begins to escape its orbit. We can see that as k=1.6, the moon totally escapes away.

Q4 a 9/10
b 8/10

File checklist for folder Q4**:**

- AbstractODESolver.cpp, AbstractODESolver.hpp
- Driver.cpp
- ForwardEulerSolver.cpp, ForwardEulerSolver.hpp
- NBodyODE.cpp, NBodyODE.hpp
- ODEInterface.cpp, ODEInterface.hpp
- StoermerVerletSolver.cpp, StoermerVerletSolver.hpp
- SymplecticEulerSolver.cpp, SymplecticEulerSolver.hpp
- Vector.cpp, Vector.hpp
- For any additional files, provide a README.txt

* Enter your output here (display only selected output if necessary).

%%%%%%%%%%%%%%%%%% Output of Driver.cpp %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
In (b) i
After one period, the state of earth is (86896.3,2.85463e+07,0) and the state of moon is (3.77332e+08,9.32929e+07,0)
In my save output:
2.37193e+06 86896.3 2.85463e+07 0 3.77332e+08 9.32929e+07 0

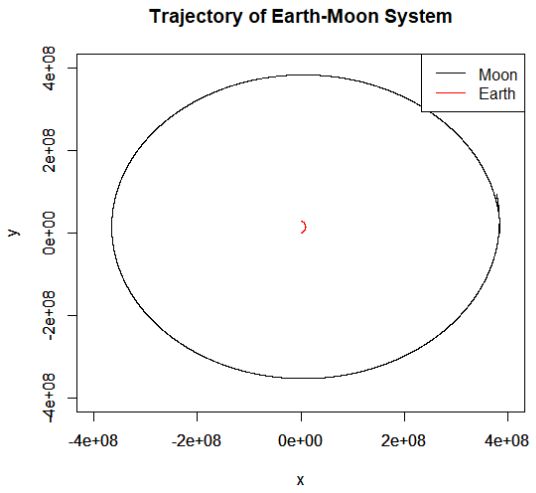
X

There aren't correct but I haven't found the bug.

In(b)iii
N=2 Time elapsed: 327 milliseconds
N=4 Time elapsed:1562 milliseconds
N=8 Time elapsed:3607 milliseconds
N=16 Time elapsed:14215 milliseconds
N=32 Time elapsed:53805 milliseconds
N=64 Time elapsed:213461 milliseconds

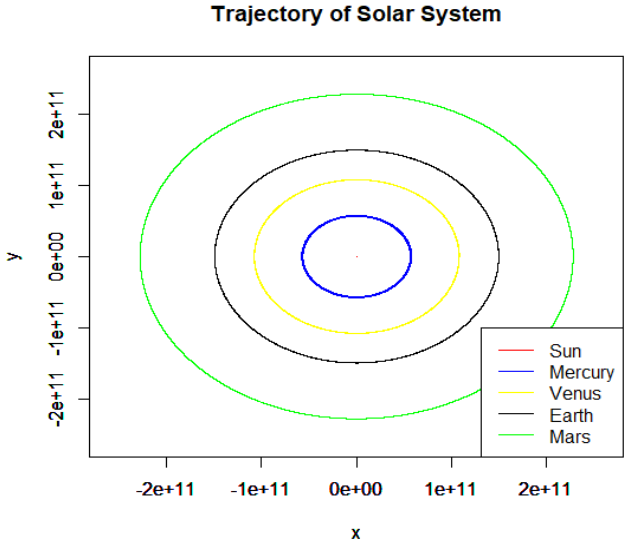
%%%

* Include your plots and comments here.
(b)i



The trajectory of the moon is approximately the same as the trajectory I got in Q3, and in this time Earth has a very small motion. Hence if we allow the earth to move, it would have little influence on the whole system so that in average time we can assume that earth is fixed.
Compare the final state of the moon after T0 with Q3, the moon pass through the initial state a little bit. It may because that the move of the earth influence the period of the moon a little bit.

(b)ii



In this question I use the solar system and I only use the data of the sun and 4 inner planets which is {Sun, Mercury, Venus, Earth, Mars}. Since I only want to plot the trajectory of the planets and there are many other factors affect the stability of the system such as the gravitation of other planets in solar system which may cause the collision, I consider each planet as a point mass (radius=0). Here I choose to use the period of Mars' motion because it is the longest among these planets so that I can plot the complete trajectory of these planets.
The plot above is the trajectories of the planets. We all know the trajectory of the planets of the solar system and my result shows that all the planets' trajectories (except Sun) are approximately circles and the Sun's motion can be ignored. Hence my result fits with the fact which indicates that my algorithm is correct and good.
Here is the initial data I used: Name={Sun, Mercury, Venus, Earth, Mars}
Mass={ 1.9891e30,3.301e23,4.8675e24,5.972e24,6.417e23}
Radius={0,0,0,0,0}
Initial Sate={(0,0,0),(5.791e10,0,0),(1.082e11,0,0),(1.496e11,0,0),(2.2794e11,0,0)}
Initial Velocity={(0,0,0),(0,4.787e4,0),(0,3.500e4,0),(0,2.978e4,0),(0,2.413e4,0) }
T_Mars=5.94e7, t0=0
h=T_Mars/10e5

Did you check the periods of other orbits?

(b)iii

In this question I also consider the bodies as point masses because I need to avoid collision happens so that I can compute the time completely. Here is the initial data I use：

N={2,4,8,16,32,64}

for(int i=0;i<N;i++)

Mass[i]=1e10

Radius[i]=0

Initial State=(i*(1e8),0,0)

Initial Velocity=(0,i*(1e6),0)

T=1e6, t0=0

h=T/1e5

I have put my results in the output part above. We can see that as N=2^k becomes N=2^(k+1), the cpu time of N=2^(k+1) is approximately 4 times of N=2^k (except that from N=4 to N=8). Hence I estimate that as N=2^37, the cpu time should be 213461*4^(37-6)=9.8441511e23 milliseconds

✓ Close enough.

Total: 90/100