Mathematics Internal Assessment

Predicting Planetary Trajectory with Euler's Method and Runge-Kutta 4

2023

Contents

1 Introduction and Rationale

Orbit determination is crucial in the fields of Astronomy for a variety of reasons, one of the most important of which is in the prevention from space-based threats. For instance, accurate predictions of Earth's orbit and the trajectories of nearby asteroids can help determine the dates, times, location, and severity of asteroid impacts. In turn, these predictions inform governments and the public to devise methods of preventing and mitigating the impact and damages. For this reason, developing precise, accurate and cost-effective mathematical methods to predict astronomical trajectories is of great importance.

Through a previous Machine Learning and Astrodynamics project, I learnt of the Runge-Kutta methods of numerical integration. They are a family of integrators developed in 1901 by two mathematicians, Carl Runge and Martin Kutta, and are still in use to this day (Christopher). Runge-Kutta 4 (henceforth referred to as RK4) in particular is most commonly used, providing a higher accuracy and faster convergence to the true values, while maintaining similar computational costs, compared to the Euler Method (henceforth referred to as Euler). This motivated me to investigate how well these two mathematical methods perform when applied to the context of predicting planet trajectories, as depicted in Fig 1.

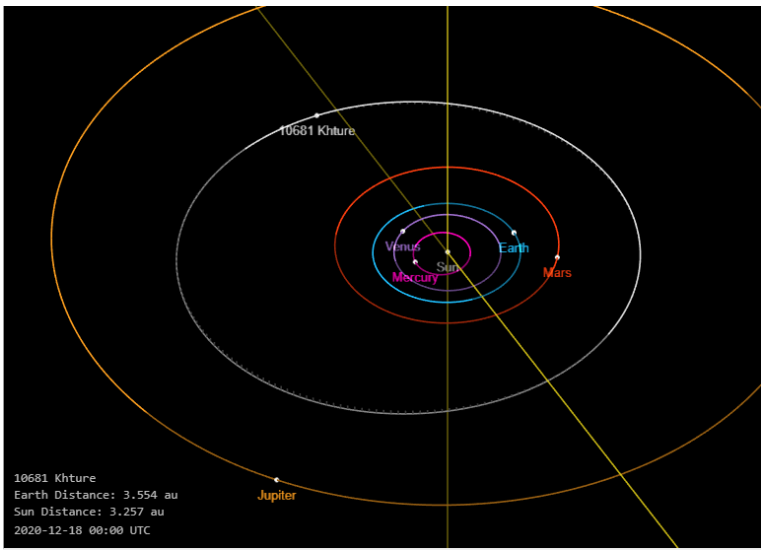


Figure 1: Computational visualization of planetary orbits in the solar system (Galkinletter)

1.1 Aim

The aim of this investigation is to analyse and compare the accuracy of Euler and RK4 method in predicting the trajectory of Mercury's orbit through a range of error analysis methods, including Absolute Error, Square Weighted Error, and 3D visualization.

2 Methodology

2.1 Data Extraction

The ground truth data is extracted from NASA JPL's Horizons system, which contains the observational ephemeris data of the orbits of over 1 million astronomical bodies. In python, I accessed the Horizons API and retrieved data as specified by the following parameters (Appendix 6.2, "Data Extraction"):

- **Reference System**: "ICRF". This ensures the plane spanned by the x,y-axis is coincident with the solar system's plane of rotation, thus results in more convenient state vectors.
- **Center Body**: "@10". This sets the Sun's center of mass (barycenter) as the coordinate center, thus its position, velocity, and acceleration are 0-vectors throughout.
- **Target Body**: 1. This sets Mercury's barycenter as the target.
- **Start Time,** T_I: "2000-JAN-01". The first piece of data extracted is Mercury's position and velocity at 2000-JAN-01, 00:00:00.
- **Stop Time,** T_F: "20YY-MMM-DD". The last piece of data extracted is Mercury's position and velocity at 20YY-MMM-DD, 00:00:00.
- **Step Size,** h: "hunits". Between 2020-JAN-01, 00:00:00 and 20YY-MMM-DD, 00:00:00, Horizons will obtain data at intervals of h units.

T_F and h are variable to extract data that spans various durations and time-steps. The

data undergoes additional formatting and processing, and it ultimately forms a time-series of $N+1 = \frac{T_F - T_I}{h}$ data points, where each data point are 6-dim state vectors, consisting of 3-dim of position and 3-dim of velocity, all in units of meters and seconds. The time-series index will be denoted with the subscript n, where $0 \leq n \leq N$. Hence the time at index n is given by $t_n = hn$. Also note that Mercury's position, velocity, and acceleration, are expressed as vectors and denoted as follows:

$$\vec{s} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \vec{v} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \quad \vec{a} = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}$$

The ground truth data at t_0 serves as the initial value for the two numerical integration methods discussed. With same h and N values as above, predicted values of position and velocities are computed based on this initial value through the python implementation of Euler and RK4. The final output in both cases is a time-series of N state vectors, which corresponds to the predicted Mercury trajectory taken at time intervals of h secs, which aligns with the ground truth data. A sample of the ground truth data is shown in Fig 2. Mathematical constructions and formulations involved in RK4 and Euler Method are described in the 3 following sections.

	X	Y	Z	VX	VY	VZ
0	-21052621072	-59537684064	-29619300156	36652.98704	-9538.146527	-8896.337239
3600	-20920617083	-59571870493	-29651251887	36682.53102	-9454.307863	-8854.617954
7200	-20788507077	-59605755036	-29683053358	36711.8853	-9370.432146	-8812.859202
10800	-20656291737	-59639337560	-29714704426	36741.04986	-9286.519552	-8771.061075
14400	-20523971745	-59672617933	-29746204951	36770.02469	-9202.570253	-8729.223666
⋮	⋮	⋮	⋮	⋮	⋮	⋮
61675200	12352563978	-58786578867	-32682649666	38135.63173	11428.40963	2149.649455
61678800	12489818906	-58745278519	-32674823033	38117.06983	11516.22678	2198.483149
61682400	12627006620	-58703662042	-32666820564	38098.29105	11604.0372	2247.335739
61686000	12764126341	-58661729460	-32658642193	38079.29519	11691.84065	2296.20712
61689600	12901177285	-58619480798	-32650287851	38060.08201	11779.63689	2345.097189

Figure 2: Sample of Mercury's ground truth state data extracted from Horizons API for $h = 3600$ and duration of 500 days

2.2 Calculating Acceleration

The distance between the Sun and Mercury is calculated from the position vector $\vec{s}_n$,

$$r(\vec{s}_n) = \sqrt{x_n^2 + y_n^2 + z_n^2}. \tag{1}$$

The Force on Mercury due to the Sun's Gravity is constructed as a function dependent on t_n,

$$\begin{aligned} F(\vec{s}_n) &= -G\frac{M_S M_M}{r(\vec{s}_n)^2} \\ &= -G\frac{M_S M_M}{x_n^2 + y_n^2 + z_n^2}, \end{aligned}$$

where $G = 6.674 \cdot 10^{-11}$ is the Gravitational constant, $M_S = 1989 \cdot 10^{24}$ is the Sun's mass, and $M_M = 3.302 \cdot 10^{22}$ is Mercury's mass in kg. Mass are obtained from the Horizons ephemeris data (NASA).

The acceleration at each t_n is calculated with Newton's Third Law, then multiplied with the normalized position vector, $\|\vec{s}_n\|$. Therefore

$$a(\vec{s}_n) = \frac{F(\vec{s}_n)}{M_M}\|\vec{s}_n\| \tag{2}$$

$$= -G\frac{M_S}{(x_n^2 + y_n^2 + z_n^2)^{3/2}}\vec{s}_n \tag{3}$$

Acceleration was implemented separated for each Euler and RK4 method, contained in Sections "Euler" and "RK4" of Appendix 6.2. A sample calculation for the acceleration at t_0 is as follows. Mercury's position and velocity at t_0 is

$$\vec{s}_0 = \begin{pmatrix} -21052621072 \\ -59537684064 \\ -29619300156 \end{pmatrix} \text{ and } \vec{v}_0 = \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} \tag{4}$$

Then plugging $\vec{s}_0$ into a results in,

$$\begin{aligned} \vec{a}_0 &= a(\vec{s}_0) \\ &= -\frac{(6.674 \cdot 10^{-11})(1989 \cdot 10^{24})}{((-21052621072)^2 + (-59537684064)^2 + (-29619300156)^2)^{3/2}} \begin{pmatrix} -21052621072 \\ -59537684064 \\ -29619300156 \end{pmatrix} \\ &\approx \begin{pmatrix} 0.008233406042 \\ 0.02328441319 \\ 0.01158372271 \end{pmatrix} \end{aligned} \tag{5}$$

It is later evident that these $a(\vec{s}_0)$ values are consistent with the computational data, as seen in the first row and columns "AX", "AY", "AZ" of Fig 3 and Fig 5.

It should noted that in reality, the total Force on Mercury due to gravity is not only dependent on the Sun, but also the rest of the objects within and beyond the solar system. Due to the Sun's massive size in comparison to the rest, their effects are almost negligible, hence they can be ignored in the computation of Mercury's acceleration.

2.3 Euler's Method

The Euler forward recursion formula is given as,

$$y_{n+1} = y_n + hf(t_n, y_n)$$

where $f(x_n, y_n) = \frac{\mathrm{d}y}{\mathrm{d}t}$ (Gottlieb). It requires the initial value $y(t_0)$ and $\frac{\mathrm{d}y}{\mathrm{d}t}$ to be known.

However, planetary orbits and trajectories are generally modelled on 2nd-order differential equations (DEs), ie. relating position to acceleration via the acceleration function in Eq 3. Therefore, as Euler only applies for 1st-order DEs, it is a necessity to decompose the relationship between position s, velocity v, and acceleration a, as follows,

$$\frac{\mathrm{d}s}{\mathrm{d}t} = s' = v \tag{6}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = v' = a \tag{7}$$

Therefore, for Eqs 6 and 7, the initial values are all known, which are $s(t_0) = \vec{s}_0$ and $v(t_0) = \vec{v}_0$ respectively. The functions $\frac{\mathrm{d}s}{\mathrm{d}t}(t_n) = \vec{v}_n$ and $\frac{\mathrm{d}v}{\mathrm{d}t}(t_n) = a(\vec{s}_n)$ where a is Eq 3 are also known at every time-index n. Thus, Euler can be applied to find the positions.

This results in the following set of equations,

$$\vec{s}_{n+1} = \vec{s}_n + h\vec{v}_n \tag{8}$$

$$\vec{v}_{n+1} = \vec{v}_n + ha(\vec{s}_n) \tag{9}$$

For a given time t_n, these equations ultimately provide the position and velocity values for t_{n+1}. A sample calculation of the formula for finding Mercury's state at time t_1 based on t_0 is

as follows. Plugging the values of $\vec{s}_0$, $\vec{v}_0$, and $\vec{a}_0$ (Eqs 4, 5) into Eqs 8 and 9 with a time-step $h = 3600$ results in Eqs 10 and 11. Again, note that $\vec{s}_1$ and $\vec{v}_1$ aligns with the computational data in row 2 of Fig 3.

$$\begin{aligned}
\vec{s}_1 &= \vec{s}_0 + 3600\vec{v}_0 \\
&= \begin{pmatrix} -21052621072 \\ -59537684064 \\ -29619300156 \end{pmatrix} + 3600 \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} \\
&\approx \begin{pmatrix} -20920616967 \\ -59571870508 \\ -29651251908 \end{pmatrix}
\end{aligned} \tag{10}$$

$$\begin{aligned}
\vec{v}_1 &= \vec{v}_0 + 3600a(\vec{s}_0) \\
&= \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} + 3600 \begin{pmatrix} 0.008233406042 \\ 0.02328441319 \\ 0.01158372271 \end{pmatrix} \\
&\approx \begin{pmatrix} 36682.6273 \\ -9454.32264 \\ -8854.635837 \end{pmatrix}
\end{aligned} \tag{11}$$

This recursion process is be iterated $N + 1$ times, starting from the initial index t_0, and until the positions and velocities at t_N is obtained. Therefore, it outputs a time-series of positions and velocities from t_0 to t_N. The python implementation for Euler is contained in the Section "Euler" of Appendix 6.2. A sample of the computed data is shown in Fig 3.

	X	Y	Z	VX	VY	VZ	r	F	AX	AY	AZ
0	-21052621072	-59537684064	-29619300156	36652.98704	-9538.146527	-8896.337239	69751355682	-9.01E+20	0.008233406042	0.02328441319	0.01158372271
3600	-20920616967	-59571870508	-29651251908	36682.6273	-9454.32264	-8854.635837	69754402799	-9.01E+20	0.008180708775	0.02329473001	0.011594699
7200	-20788506497	-59605755120	-29683053463	36712.07786	-9370.461612	-8812.894921	69757377450	-9.01E+20	0.008128008969	0.02330499847	0.01160564973
10800	-20656290348	-59639337765	-29714704680	36741.33869	-9286.563617	-8771.114582	69760279631	-9.01E+20	0.008075306408	0.02331521867	0.01161657495
14400	-20523969200	-59672618312	-29746205417	36770.40979	-9202.62883	-8729.294912	69763109339	-9.00E+20	0.008022600878	0.02332539069	0.01162747475
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
61675200	54438622442	-35938678052	-24843107635	18420.73971	34539.66687	16539.27978	69802093024	-8.99E+20	-0.02124384433	0.01402452244	0.009694644861
61678800	54504799445	-35814244373	-24783503407	18344.26187	34590.15515	16574.1805	69768583930	-9.00E+20	-0.02130033039	0.01399611127	0.009685327095
61682400	54570700762	-35689629119	-24723773596	18267.58068	34640.54115	16609.04768	69735041329	-9.01E+20	-0.02135687284	0.01396754779	0.009675933815
61686000	54636325660	-35564832662	-24663918324	18190.69594	34690.82433	16643.88104	69701465343	-9.02E+20	-0.02141347145	0.01393883135	0.009666464654
61689600	54701673406	-35439855370	-24603937714	18113.60744	34741.00412	16678.68031	69667856096				

Figure 3: Sample of trajectory information computed based on Euler for $h = 3600$ and duration of 500 days

2.4 RK4

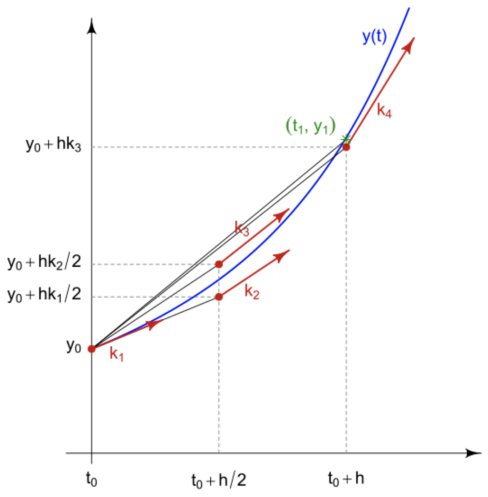


Figure 4: Visualization of RK4 (HilberTraum)

Euler and RK4 is fundamentally different as Euler fails to take into account the curvature of the solution (Barker). It only considers the slope at the lower bound of a time-step. However, RK4 accounts for four slopes taken at the lower bound, mid-point, and upper bound as shown in Fig 4, then computes the next step with weighted averages of these slopes.

Like Euler, RK4 method is only suitable for 1st-order DEs, as it only considers the first derivatives in its formulation. However, although the 2nd-order alternative of Euler is relatively simple, the alternate RK4 is more convoluted due to additional complexity in considering midpoints of the time-step.

The following construction of RK4 for the Mercury problem is adapted based on the general RK4 formulation for 2nd-order DEs, detailed in (Voesenek). Fundamentally, it utilizes the same decomposition of the 2nd-order DEs in Eqs 6 and 7. Two RK4 equations corresponding to the solution of each DE are,

$$\vec{s}_{n+1} = \vec{s}_n + \frac{h}{6}(\vec{k}_{1_{s_{n+1}}} + 2\vec{k}_{2_{s_{n+1}}} + 2\vec{k}_{3_{s_{n+1}}} + \vec{k}_{4_{s_{n+1}}}) \tag{12}$$

$$\vec{v}_{n+1} = \vec{v}_n + \frac{h}{6}(\vec{k}_{1_{v_{n+1}}} + 2\vec{k}_{2_{v_{n+1}}} + 2\vec{k}_{3_{v_{n+1}}} + \vec{k}_{4_{v_{n+1}}}) \tag{13}$$

The k coefficients in Eqs 12 and 13 are given as follows,

$$\vec{k}_{1_{s_{n+1}}} = \vec{v}_n \qquad \vec{k}_{1_{v_{n+1}}} = a\left(\vec{s}_n\right)$$

$$\vec{k}_{2_{s_{n+1}}} = \vec{v}_n + \frac{h}{2}\vec{k}_{1_{v_{n+1}}} \qquad \vec{k}_{2_{v_{n+1}}} = a\left(\vec{s}_n + \frac{h}{2}\vec{k}_{1_{s_{n+1}}}\right)$$

$$\vec{k}_{3_{s_{n+1}}} = \vec{v}_n + \frac{h}{2}\vec{k}_{2_{v_{n+1}}} \qquad \vec{k}_{3_{v_{n+1}}} = a\left(\vec{s}_n + \frac{h}{2}\vec{k}_{2_{s_{n+1}}}\right)$$

$$\vec{k}_{4_{s_{n+1}}} = \vec{v}_n + h\vec{k}_{3_{v_{n+1}}} \qquad \vec{k}_{4_{v_{n+1}}} = a\left(\vec{s}_n + h\vec{k}_{3_{s_{n+1}}}\right)$$

	X	Y	Z	VX	VY	VZ	r	F	AX	AY	AZ
0	-21052621072	-59537684064	-29619300156	36652.98704	-9538.146527	-8896.337239	69751355682	-9.01E+20	0.008233406042	0.02328441319	0.01158372271
3600	-20920617080	-59571870486	-29651251884	36682.53245	-9454.304055	-8854.616072	69754402804	-9.01E+20	0.008180708818	0.02329472999	0.01159469899
7200	-20788507066	-59605755009	-29683053344	36711.88814	-9370.42453	-8812.855437	69757377474	-9.01E+20	0.008128009183	0.02330499841	0.01160564967
10800	-20656291713	-59639337498	-29714704396	36741.05411	-9286.508125	-8771.055425	69760279686	-9.01E+20	0.008075306923	0.02331521851	0.01161657482
14400	-20523971705	-59672617823	-29746204896	36770.03035	-9202.555014	-8729.216129	69763109436	-9.00E+20	0.008022601823	0.0233253904	0.0116274745
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
61675200	12513191648	-58736581984	-32672623957	38114.52694	11533.42129	2207.705	68367144043	-9.38E+20	-0.00519705246	0.0243948232	0.01356978663
61678800	12650370137	-58694903593	-32664588275	38095.70904	11621.23923	2256.56566	68352753416	-9.38E+20	-0.00525734540	0.02439291326	0.01357501963
61682400	12787480492	-58652909070	-32656376662	38076.67397	11709.05018	2305.445107	68338293338	-9.38E+20	-0.00531770103	0.02439093732	0.01358022388
61686000	12924521929	-58610598440	-32647989048	38057.42152	11796.8539	2354.343237	68323763842	-9.39E+20	-0.00537811962	0.02438889512	0.01358539926
61689600	13061493666	-58567971731	-32639425368	38037.95144	11884.65015	2403.259947	68309164962				

Figure 5: Sample of trajectory information computed based on RK4 for $h = 3600$ and duration of 500 days

Due to the 2nd order nature of the problem, the k coefficients will be computed alternately, starting with $\vec{k}_{1_{s_{n+1}}}$ and $\vec{k}_{1_{v_{n+1}}}$, and so on until $\vec{k}_{4_{s_{n+1}}}$ and $\vec{k}_{4_{v_{n+1}}}$. Once these k coefficients are fully calculated for one time-index n, Eqs 12 and 13 can be found. A sample calculation of the formula for calculating $\vec{s}_1$ and $\vec{v}_1$ through RK4 is given in Appendix 6.1.

Like Euler's Method, this process will be repeated N times, until all the positions and velocities are computed for t_0 to t_N. The python implementation for Euler is contained in the Section "RK4" of Appendix 6.2. A sample of the computed data is shown in Fig 5.

One limitation of this investigation is the uncertainty around the code implementation. As I programmed all the processes from scratch, I did not do significant research into optimizing the code, particularly for Euler and RK4 implementation. This means I lack the ability to evaluate the computational costs associated, such as the GPU runtime and memory required for each method. This limits the depth and breadth of comparing between the efficiency of the two methods, since only the accuracy can be considered, but not computational efficiency.

2.5 Accuracy Analysis

In order to critically analyse and compare the results of Euler and RK4, the error analysis methods described below are used, including details of relevant equations and sample calculations. Note that A, E, R denotes ground truth, Euler, and RK4 data respectively.

1. **Magnitude Computation and Visualization**

The magnitudes of position and velocity vectors of A, E, R are computed with the distance formula r from Eq 1, then plotted simultaneously. This helps identify potential deviations of E and R from A, particularly in the orbital periods. This is further explored in Section 3.

2. **Square Weighted Error (SWE)**

SWE is calculated based on the discrepancy of the magnitudes of their position and velocity vectors of E and R against A. In particular, for a vector belong to E or R, $\vec{er}$, and its corresponding vector in A, $\vec{a}$,

$$SWE = \left(\frac{r(\vec{er})}{r(\vec{a})}\right)^2.$$

A sample SWE calculation for the Euler position for time-step $h = 3600$, at $n = 1, t = 3600$ is shown as follows.

$$\begin{aligned} SWE &= \left(\frac{r(\vec{s}_{1E})}{r(\vec{s}_{1A})}\right)^2 \\ &= \left(\frac{r\begin{pmatrix}-20920616967\\-59571870508\\-29651251908\end{pmatrix}}{r\begin{pmatrix}-20920617083\\-59571870493\\-29651251887\end{pmatrix}}\right)^2 \\ &= \left(\frac{69754402799}{69754402812}\right)^2 \\ &= 0.9999999996 \end{aligned}$$

SWE is the main indicator for the growth in deviation between indicates deviations between the orbital periods of A against E or R trajectories. The sum of all SWE values across a time-step is also computed.

3. **3D Visualization**

Through the Plotly library, the position vectors of A, E, R are simultaneously plotted in 3D (Plotly). This aids in the intuitive interpretation of the differences in errors and

accuracy between the Euler and RK4.

The Python implementations for these methods are contained in Sections "SWE", "Magnitude Visualization", "3D Visualization" of Appendix 6.2 respectively. It should be noted that a few other methods of quantifying discrepancy was considered. However, I felt that those listed above were sufficient in providing a nuanced understanding of the discrepancy and exploration of the underlying reasons. That said, other error analysis methods can be utilized in extension to this investigation.

3 Results and Analysis

Four different scenarios with time-steps, $h = 1sec, 1hour, 1day, 5days,$ and $10days$ ($h = 6, 3.6 \cdot 10^3, 8.64 \cdot 10^5, 4.32 \cdot 10^6, 8.64 \cdot 10^6 secs$) were simulated for a total duration of 800 days (2000-JAN-01 to 2002-MAR-11). However, due to the significant cost in terms of processing and time involved in generating 800 days of trajectory data for 1 sec time-steps, therefore, the total duration for $h = 1sec$ is set to 1 day (2000-JAN-01 to 2000-JAN-02). These parameters are specified in Section "Input Parameters" of Appendix 6.2. All the error plots and images of the 3D simulation have been added in Appendix 6.3, while significant observations are detailed in the following subsections.

3.1 Overall accuracy of Euler vs RK4

The position SWE plots and 3D simulations of Euler and RK4 for each time-step both suggest that in general, RK4 constructs a more accurate trajectory. For $h = 1sec$, there is virtually no difference between the errors in the SWE plot (Fig 6).

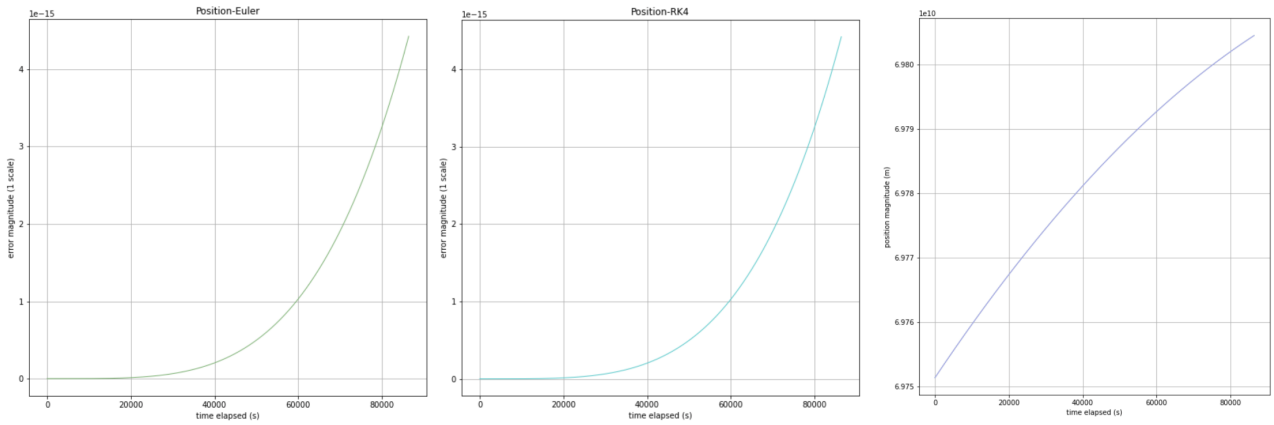


Figure 6: SWE plot (Graph 1, 2) and Magnitude Plot (Graph 3) of Euler and RK4 for $h = 1sec$.

Thus taking a sum over the 86,400 SWE error data points for position, we obtained a value of $7.6501 \cdot 10^{-11}$ and $7.6446 \cdot 10^{-11}$ for Euler and RK4 respectively, with a difference of $5.4965 \cdot 10^{-14}$. Therefore, this highlights there are very minute differences between the two methods. The magnitude graph further illustrates there is virtually no phase stretching, thus the orbital periods also remains the same (Fig 6).

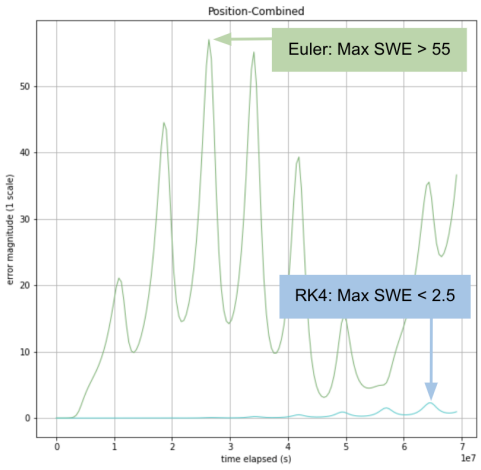


Figure 7: SWE plot of Euler (Green) and RK4 (Blue) for $h = 5days$.

For $h = 1hour, 1day, 5days$, the magnitude of SWE of RK4 pales in comparison to Euler. For instance, at $h = 5days$, the maximum error is > 55 and < 2.5 for Euler and RK4 respectively (Fig 7).

Moreover, across all h, as the time-step grows, the phase stretch of Euler magnitudes increases dramatically. It is evident in Fig 8 that the time elapsed at which the minimum and maximum position magnitudes occur increases in relation to the original orbit for Euler. The degree of discrepancy is much greater at $h = 5days$ than it is at $h = 1hour$. On the other hand, the RK4 magnitudes remain fairly accurate with true data. This discrepancy also increases with the greater time-step of $h = 5days$, but is minimal in comparison to Euler.

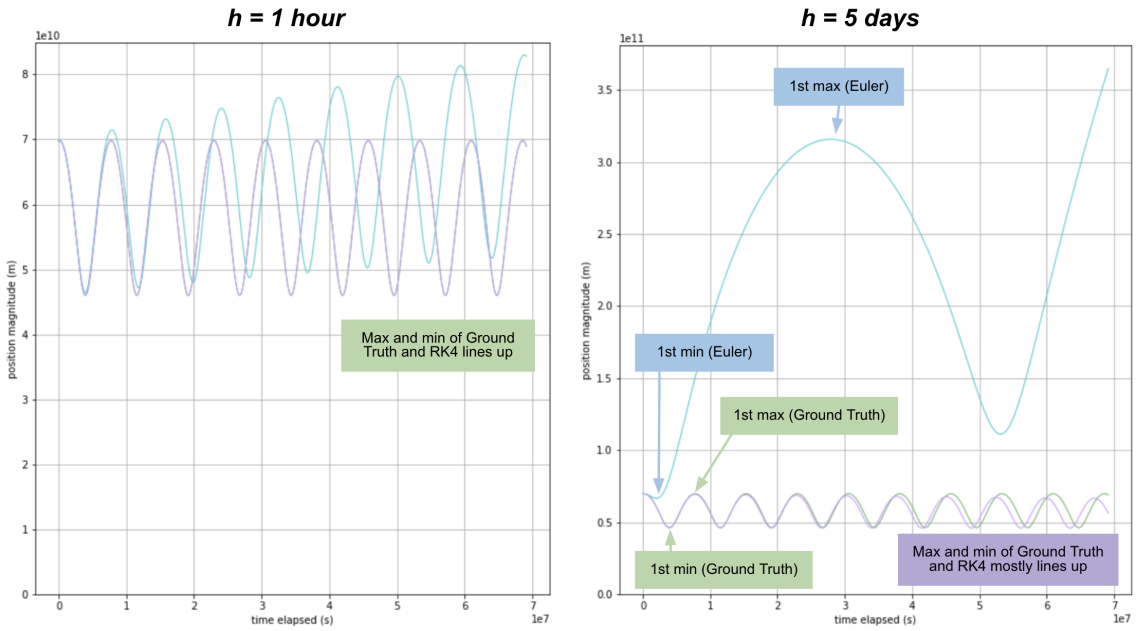


Figure 8: Magnitude plot of Ground Truth (Green), Euler (Blue), and RK4 (Purple) for $h = 1hour$ and $h = 5days$.

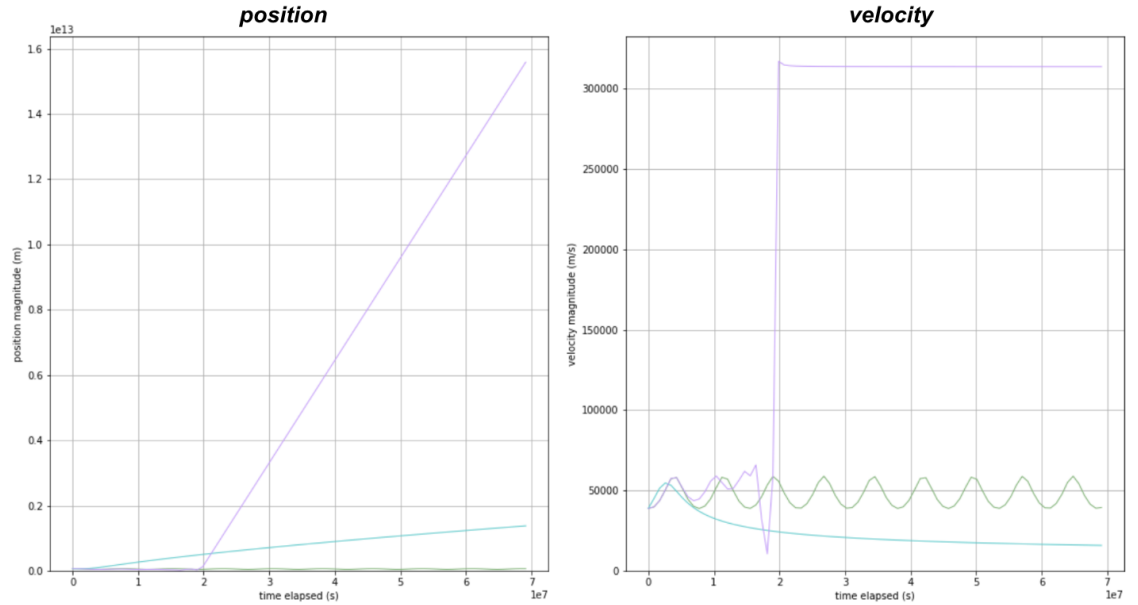


Figure 9: Position and Velocity Magnitude Plots of Ground Truth (Green), Euler (Blue), and RK4 (Purple) for $h = 10days$.

However, the trend of Euler's greater inaccuracy changes for $h = 10days$. Apparent in both magnitude plots in Fig 9, although Euler errors start to increase at an earlier elapsed time between $0 < t < 1 \cdot 10^7$, the RK4 errors start to grow between $2 \cdot 10^7 < t < 3 \cdot 10^7$ to a much higher discrepancy. At the end of the 500 days, the RK4 errors completely overshadow Euler,

in stark contrast to the previous trends.

These discrepancies can also be qualitatively contextualized with the 3D simulations in Fig 10. With $h = 1sec$, there is virtually no difference between the 3 trajectories. There are distinct deviations in the Euler trajectory at $h = 1hour, 1day$, although the orbits are still relatively consistent with the original orbit, while the RK4 trajectory is virtually the same. We begin seeing minor deviations in the RK4 trajectory for $h = 5days$, and the Euler trajectory becomes much less consistent with the original. Finally, the $h = 10days$ visualization clearly illustrates the massive RK4 deviation. Euler was able to retain a discernible curvature in its path, whereas RK4 essentially veers off the orbit entirely. These observations are consistent with the quantitative data above.

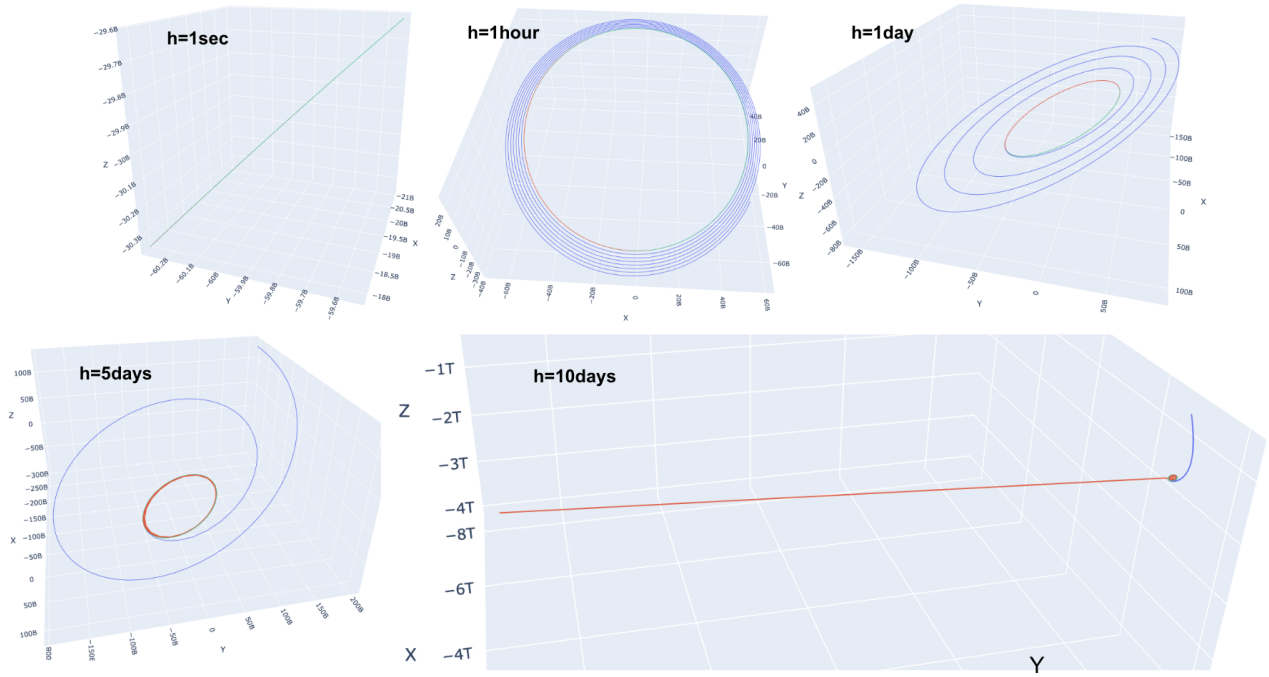


Figure 10: 3D Visualizations of Ground Truth (Green), Euler (Blue), and RK4 (Red).

The analysis suggests that for very small time-steps, there are minimal accuracy differences between Euler and RK4. As the time-step increases, the performance of these methods begin to deviate, where RK4 clearly presents as more optimal. However, there exists a breaking point with larger time-steps, where both methods are no longer consistent with reality, and RK4 may see greater inaccuracy than Euler.

3.2 Differences in deviation behaviour between Euler and RK4 for larger time-steps

By analysing the 3D visualizations for $h = 5days$ in Fig 11, there is a clear difference between the way Euler and RK4 trajectories deviates. Euler gradually veers off the circular path of the original trajectory, growing greater in distance. The trajectory eventually veers back close to the original orbit, forming an ellipse-like shape, albeit with a significant orbital period difference as discussed above. However, it doesn't return to the original trajectory for too large time-steps.

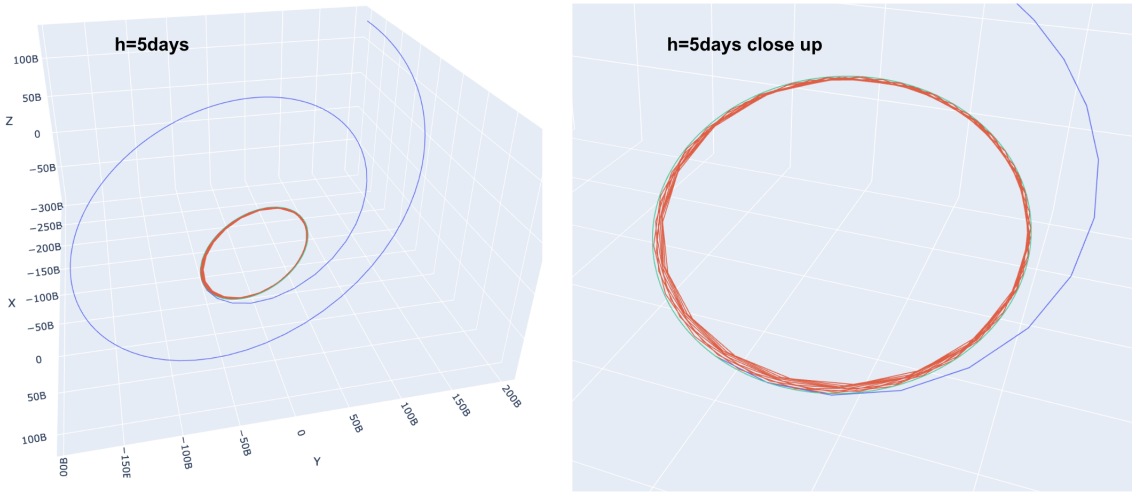


Figure 11: Close up of D Visualizations of Ground Truth (Green), Euler (Blue), and RK4 (Red) for $h = 5days$.

On the other hand, the RK4 trajectory does not deviate away from the orbit, but rather propagates within it. Upon closer inspection into the 3D simulation (Fig 11) and the magnitude plot (Fig 8) for $h = 5days$, this propagation continues throughout the 800 days, where the position magnitude gradually decreases. With a larger time-step of $h = 10days$, the magnitude decreases at a much faster rate, until it reaches a point where it starts to venture away from the original orbit completely, accumulating errors even greater than Euler as noted in the above section (Fig 9).

A plausible explanation for the difference in the behaviours of Euler and RK4 trajectories is embedded in the respective method's formulation. As discussed in the Methodology, in

Sections 2.3 and 2.4, RK4 places significant emphasis on the curvature. An extreme case is shown for a circular orbit in Fig 12, where the time-step is assumed to cover a quarter of the circle. A rough analogy of Euler and RK4 is replicated in this scenario to demonstrate how each method results in different behaviours of its predicted trajectory. When the time-step is greater relative to the orbit itself, RK4 will over-rely on the curvature, and under-predicts the trajectory, in contrast to Euler, which only relies on the slope information at the initial point and over-predicts the trajectory.

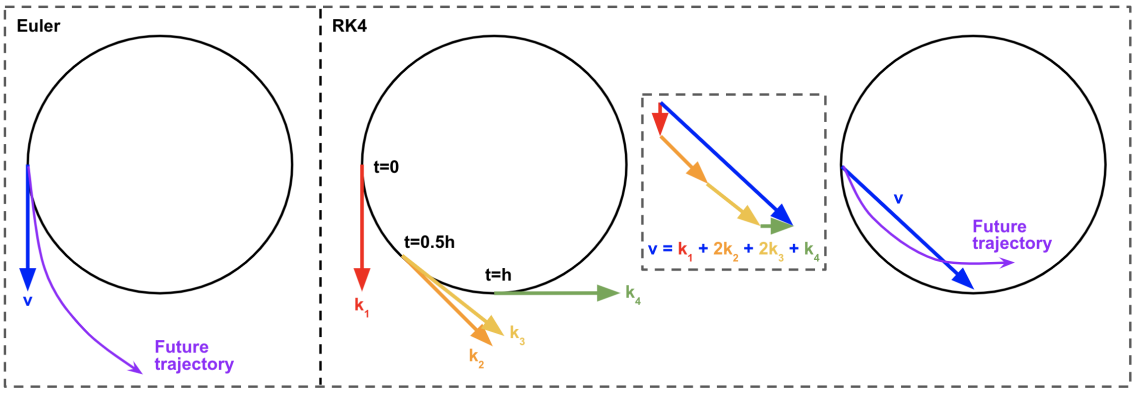


Figure 12: An analogy of Euler and RK4 methods on a single large time-step in a circular orbit scenario.

3.3 Periodic behaviour

Throughout the magnitude and SWE plots in Appendix 6.3, with the exception of $h = 1sec$, we can observe a periodic nature, sinusoidal-like behaviour to the graphs. In particular, this is consistent with elliptical orbits, as instead of orbiting around a constant radius away from the sun, the distance oscillates between the 2 values of the major and minor axis, hence creating the sinusoidal-like evolution.

The periodic behaviour eventually manifests in the speed due to the direct influence of oscillating position magnitude on the acceleration, and thus on the speed. This is evident in the magnitude plots such as Fig 8, where both the positions and velocities of the true data exhibits this sinusoidal behavior. Likewise, this has also manifested in the SWE errors, where

RK4 data in particular follows periodic pattern for $h = 1hour, 1day, 5days$ (Fig 13). Due to its relative accuracy in comparison with ground truth, it retains the neat periodic pattern, with an increasing overall trend as the deviation grows.

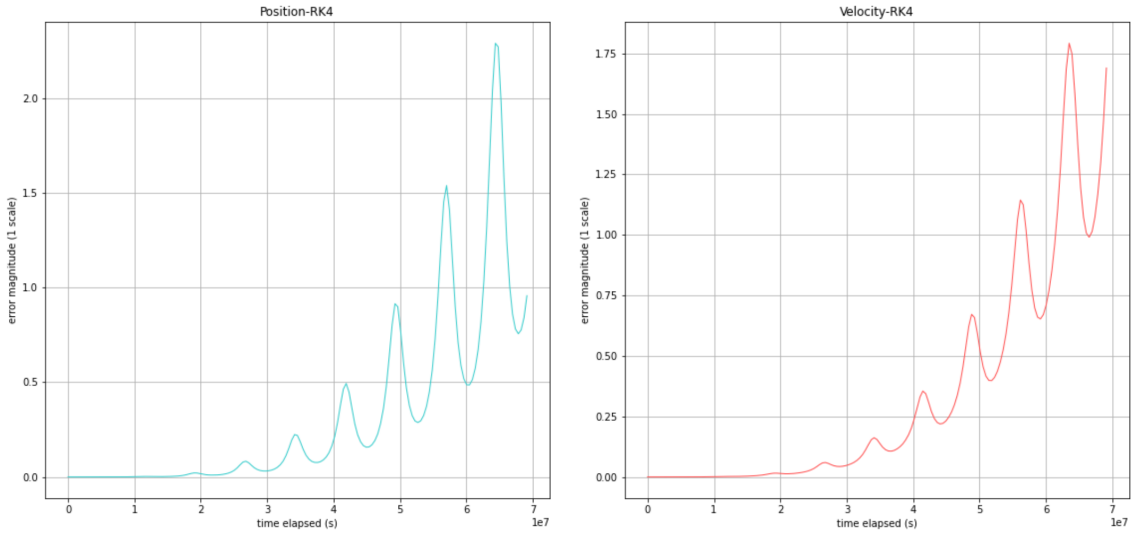


Figure 13: Increasing Sinusoidal Pattern in SWE error for $h = 5days$

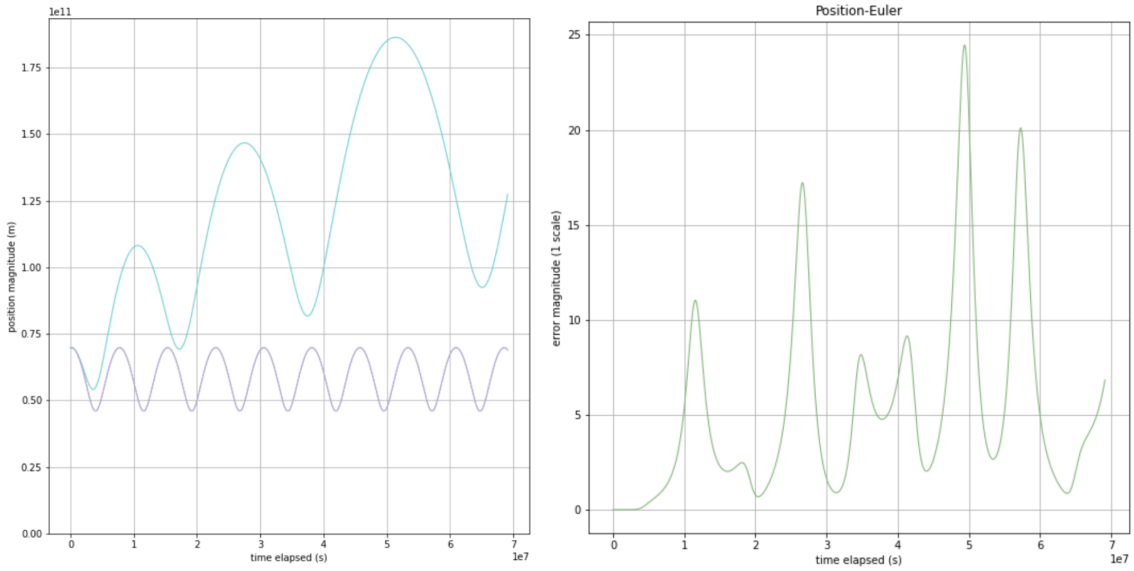


Figure 14: Magnitude (Graph 1) and SWE plot (Graph 2) of position for Euler at $h = 1day$.

On the other hand, the Euler data experiences greater fluctuation and discrepancies in this periodic pattern, seen mostly in larger time-steps and later elapsed times, which occurs mainly due to deviations in the orbital period. For example, the $h = 1day$ SWE plot for Euler position has a highly fluctuating periodic graph, where its minimum and maximums do not

monotonically increase (Fig 14). This is expected from its magnitude plot, where the orbital periods of the Euler trajectory differs significantly as discussed in Section 3.1, causing the SWE errors to fluctuate in an inconsistent manner.

3.4 Relationship between magnitudes of position and speed

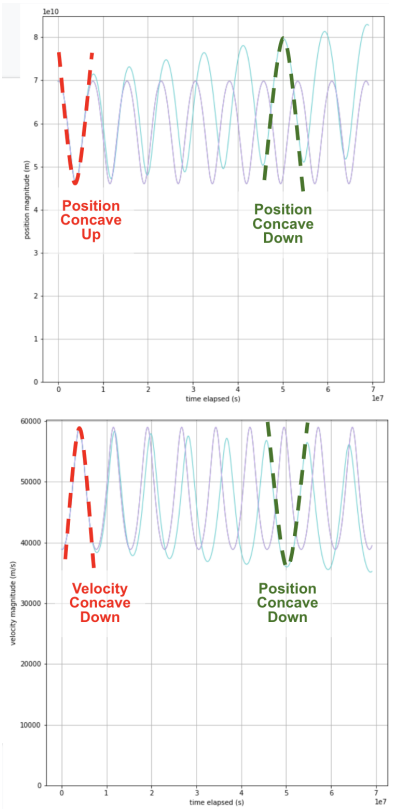


Figure 15: Magnitude plot of position and velocity for Euler at $h = 1hour$

An interesting observation I made is the apparent inverted relationship between the magnitudes of position and velocity across all magnitude plots, but most clearly depicted with the Euler data for $h = 1hour, 1day$. As seen in Fig 15, where the position graph has a concave up shape, the speed graph would be concave down and the converse also applies, and the minimum and maximums lies approximately on the same time elapsed values. Furthermore, where the overall trend of position decreases, speed will increase.

Upon further deliberation, I realized that this relationship is actually consistent with the physical dynamics of an elliptical orbit, as shown in Fig 16. As the object in an elliptical orbit approaches the position of highest magnitudes from the lowest magnitudes (A), its speed will decrease from its highest value to its lowest. Therefore, when the magnitude of position is increasing, the velocity graph is decreasing, the contrary also applies (B). This is due to the greater force of gravitational acceleration experienced at the smaller distances, resulting a greater value of velocity, and vice versa. Therefore, this suggests that the numerical trends between positions and velocities can be explained with the physical nature of elliptical orbits and gravitational fields.

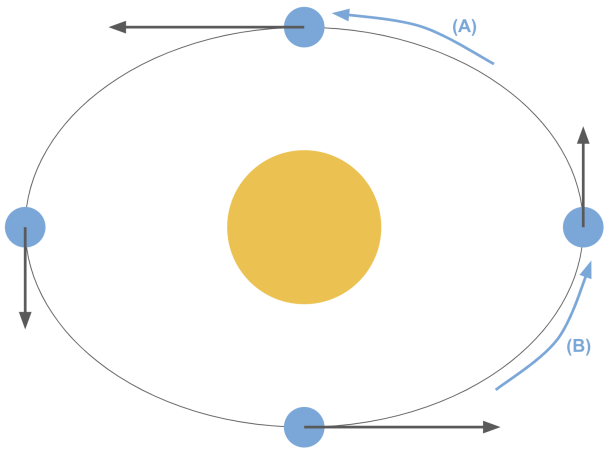


Figure 16: Velocity vectors (black) of an object orbiting around another object in an elliptical orbit. Blue vectors represent motion of the planet along a section of its orbit.

4 Conclusion

I believe that my aim was achieved, through technically implementing both Euler and RK4 in Python, running the simulations and producing trajectory predictions for various time-step and conditions, constructing error analysis methods to evaluate the accuracy of computed and true trajectories, then conducting and in-depth analysis of the quantitative data found, to finally determine the most accurate computational method. All things considered, RK4 outperforms Euler in accurately simulating the trajectory of Mercury's orbit over a duration of several hundred days, within the same time-step conditions that ranges between $1hour \leq h \leq 5days$. It is able to navigate around mercury's slightly elliptical orbit, and maintain a consistent orbital period, only experiencing minor deviations as its time-steps approaches the upper bound of this range. For very small time-steps, in this case $h = 1sec$, there is minimal difference in the accuracy of these two methods.

This conclusion is supported by the differences in Local and Global Truncation Error (LTE, GTE). According to Berg, the former is the error made at a single step of integration, while the latter is the aggregated error across all steps. The LTE and GTE are given as Euler: $O(h^2), O(h)$ and RK4: $O(h^5), O(h^4)$, and obtain based on the derivation of each method from

the Taylor Expansion (Gottlieb, Berg). Generally, the higher the order of error, the more accurate. Therefore, it is sensible for RK4 to outperform Euler in accuracy. However, as pointed out prior, there is a limit to RK4's better performance, where for large time-steps

In terms of future predictions, RK4 is the more powerful method, offering much more accurate results for larger time-steps. As discussed prior, there is a certain upper limit of time-step in which RK4 starts to fail and significantly under-perform. Therefore it is important to be wary of such limits and conduct simulations to determine their values, which vary depending on the situation. However, it is more plausible that smaller time-steps will be used for this purpose. In this case, both RK4 and Euler are strong contenders in predicting orbital trajectories. This is particularly true for short term predictions, where deviations in Euler will not be as apparent. However, as briefly addressed in Section 2.4, an extension in analyzing the computing efficiency, both in Big O complexity and in data storage and memory can be conducted to make the final determination of the best method in highly accurate time-steps. This will require an in-depth investigation of the theory behind Euler and RK4 derivations, their truncated errors, and optimizing the code implementations.

In hindsight, the choice of h parameters is also a limitation of the investigation. 5 h values were studied, which have provided interesting insights, but the precise relationship between the time-step and efficiency is sparse and disjointed. An improvement can be made by considering a greater amount of time-steps, then graphing the h parameters against the aggregated errors, for instance, total SWE error. Note that this should be done on a log scale, to fit data ranging from 1sec to 864000secs in a single plot. One such benefit of this is the ability to identify the time-step at which RK4 begins to fail and under-perform. Other further extensions can look into the accuracy differences within the Runge-Kutta family of methods, applying Euler and RK4 to more complicated solutions, and investigating the factors that may influence the RK4 failure benchmark.

5 Works Cited

Barker, Christopher A. "Numerical Methods for Solving Differential Equations - The Runge-Kutta Method". 2017. spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf.

Berg, Ingo. "Runge Kutta vs Euler". Mar. 2015. beltoforion.de/en/runge-kutta_vs_euler/.

Christopher, Kelsa. Math 373 Runge-Kutta. Apr. 2004. showard . sdsmt . edu / Math373 / _ AppliedNumMethodsText_SMH/RK_History.htm.

Galkinletter. File: 10681(with Jupiter).png. Dec. 2020. commons . wikimedia . org / wiki / File: %D0%9E%D1%80%D0%B1%D0%B8%D1%82%D0%B0_%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%BE%D0%B8%D0%B4%D0%B0_10681(with_Jupiter).png.

Gottlieb, Sigal. "Euler's Method, Taylor Series Method, Runge Kutta Methods, Multi-Step Methods and Stability." www.cfm.brown.edu/people/sg/AM35odes.pdf.

HilberTraum. Slopes used by the classical Runge-Kutta method. Nov. 2017. commons.wikimedia.org/wiki/File:Runge-Kutta_slopes.svg.

NASA. Horizon API. 2022. ssd-api.jpl.nasa.gov/doc/horizons.html.

Plotly. Plotly Open Source Graphing Library for Python. 2023. plotly.com/python/.

Voesenek, C.J. "Implementing a Fourth Order Runge-Kutta Method for Orbit Simulation". June 2008. spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf.

6 Appendix

6.1 RK4 Sample Calculation

Sample Calculations for $h = 3600$ are be shown in detail for finding $\vec{k}_{1_{s_1}}, \vec{k}_{1_{v_1}}$ and $\vec{k}_{2_{s_1}}, \vec{k}_{2_{v_1}}$.

$$\begin{aligned}
\vec{k}_{1_{s_1}} &= \vec{v}_0 \\
&= \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} \\
\vec{k}_{1_{v_1}} &= a\left(\vec{s}_0\right) \\
&= \begin{pmatrix} 0.008233406042 \\ 0.02328441319 \\ 0.01158372271 \end{pmatrix} \\
\vec{k}_{2_{s_1}} &= \vec{v}_0 + \frac{h}{2}\vec{k}_{1_{v_1}} \\
&= \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} + \frac{3600}{2}\begin{pmatrix} 0.008233406042 \\ 0.02328441319 \\ 0.01158372271 \end{pmatrix} \\
&= \begin{pmatrix} 36667.80717 \\ -9496.234584 \\ -8875.486538 \end{pmatrix} \\
\vec{k}_{2_{v_1}} &= a\left(\vec{s}_0 + \frac{h}{2}\vec{k}_{1_{s_1}}\right) \\
&= a\left(\begin{pmatrix} -21052621072 \\ -59537684064 \\ -29619300156 \end{pmatrix} + \frac{3600}{2}\begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix}\right) \\
&= a\begin{pmatrix} -20986645695 \\ -59554852728 \\ -29635313563 \end{pmatrix} \\
&= -\frac{(6.674 \cdot 10^{-11})(1989 \cdot 10^{24})}{((-20986645695)^2 + (-59554852728)^2 + (-29635313563)^2)^{3/2}}\begin{pmatrix} -20986645695 \\ -59554852728 \\ -29635313563 \end{pmatrix} \\
&= \begin{pmatrix} 0.00820704733 \\ 0.02328954813 \\ 0.01158919937 \end{pmatrix}
\end{aligned}$$

The rest of k coefficients are,

$$\vec{k}_{3_{s_1}} = \begin{pmatrix} 36667.75973 \\ -9496.225341 \\ -8875.47668 \end{pmatrix} \quad \vec{k}_{3_{v_1}} = \begin{pmatrix} 0.008207068095 \\ 0.02328960716 \\ 0.01158922875 \end{pmatrix}$$

$$\vec{k}_{4_{s_1}} = \begin{pmatrix} 36682.53249 \\ -9454.303942 \\ -8854.616015 \end{pmatrix} \quad \vec{k}_{4_{v_1}} = \begin{pmatrix} 0.008180708839 \\ 0.02329472999 \\ 0.01159469898 \end{pmatrix}$$

Hence $\vec{s}_1$ and $\vec{v}_1$ can be computed as follows,

$$\begin{aligned}
\vec{s}_1 &= \vec{s}_0 + \frac{h}{6}\left(\vec{k}_{1_{s_1}} + 2\vec{k}_{2_{s_1}} + 2\vec{k}_{3_{s_1}} + \vec{k}_{4_{s_1}}\right) \\
&= \begin{pmatrix} -21052621072 \\ -59537684064 \\ -29619300156 \end{pmatrix} + \frac{3600}{6}\left(\begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} + 2\begin{pmatrix} 36667.80717 \\ -9496.234584 \\ -8875.486538 \end{pmatrix}\right. \\
&\qquad \left. +2\begin{pmatrix} 36667.75973 \\ -9496.225341 \\ -8875.47668 \end{pmatrix} + \begin{pmatrix} 36682.53249 \\ -9454.303942 \\ -8854.616015 \end{pmatrix}\right) \\
&= \begin{pmatrix} -20920617080 \\ -59571870486 \\ -29651251884 \end{pmatrix}
\end{aligned}$$

$$\begin{aligned}
\vec{v}_1 &= \vec{v}_0 + \frac{h}{6}\left(\vec{k}_{1_{v_1}} + 2\vec{k}_{2_{v_1}} + 2\vec{k}_{3_{v_1}} + \vec{k}_{4_{v_1}}\right) \\
&= \begin{pmatrix} 36652.98704 \\ -9538.146527 \\ -8896.337239 \end{pmatrix} + \frac{3600}{6}\left(\begin{pmatrix} 0.008233406042 \\ 0.02328441319 \\ 0.01158372271 \end{pmatrix} + 2\begin{pmatrix} 0.00820704733 \\ 0.02328954813 \\ 0.01158919937 \end{pmatrix}\right. \\
&\qquad \left. +2\begin{pmatrix} 0.008207068095 \\ 0.02328960716 \\ 0.01158922875 \end{pmatrix} + \begin{pmatrix} 0.008180708839 \\ 0.02329472999 \\ 0.01159469898 \end{pmatrix}\right) \\
&= \begin{pmatrix} 36682.53245 \\ -9454.304055 \\ -8854.616072 \end{pmatrix}
\end{aligned}$$

6.2 Code

Imports

```
import requests
import os

import pandas as pd
import numpy as np
import scipy

import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True
import matplotlib.patches as mpatches
import plotly.express as px

#from google.colab import drive
#drive.mount('/content/drive')
```

Parameter

hour to seconds

500 days: 2001-MAY-15

2000-MAR-24 2000-MAR-29

```
hour = 10*24
print(hour*3600)
```

```
864000
```

Input Parameters

```
#@title Input Parameters
dt =  864000#@param {type:"integer"}
target_body = 1#@param {type:"integer"}
center_body = "@10"#@param {type:"string"}

timestep = "10day"#@param {type:"string"}
start = "2000-JAN-01"#@param {type:"string"}
stop = "2002-MAR-11"#@param {type:"string"}
duration = '800days'#@param {type:"string"}

iteration_folder = '800_10day'#@param {type:"string"}
path_name = f'/content/drive/MyDrive/mathIA/{iteration_folder}/'
first_mount = True#@param {type:"boolean"}
if first_mount == True:
  os.makedirs(f'/content/drive/MyDrive/mathIA/{iteration_folder}')

#plot_Poscomp = True#@param {type:"boolean"}
#plot_Velcomp = True#@param {type:"boolean"}
#plot_MagAngle = True#@param {type:"boolean"}
#plot_SWE = True#@param {type:"boolean"}
#plot_3d = True#@param {type:"boolean"}

show_dataframes = True#@param {type:"boolean"}
save_all = True#@param {type:"boolean"}
```

dt: 864000

target_body: 1

center_body: "@10"

timestep: "10day"

start: "2000-JAN-01"

stop: "2002-MAR-11"

duration: "800days"

iteration_folder: "800_10day"

first_mount: ☑

show_dataframes: ☑

save_all: ☑

Simulation Parameters

```
#@title Simulation Parameters
M_p = 3.302e22#@param {type:"number"}
```

M_p: 3.302e22

```
M_s = 1988500e24#@param {type:"number"}

G=6.6743e-11
multiply = G*M_s*M_p
```

M_s: 1988500e24

Functions

Data Extraction

```
def extract(target_body, center_body, timestep, start, stop, dt, save):
  def URL(target_body, center_body, timestep, start, stop):
    url = "https://ssd.jpl.nasa.gov/api/horizons.api?"\
      f"COMMAND={target_body}"\
      f"&CENTER={center_body}"\
      f"&STEP_SIZE={timestep}"\
      f"&START_TIME={start}"\
      f"&STOP_TIME={stop}"\
      "&format=text"\
      "&EPHEM_TYPE=VECTORS"\
      "&MAKE_EPHEM=YES"\
      "&OUT_UNITS=KM-S"\
      "&CSV_FORMAT=YES"\
      "&ELEM_LABELS=YES"\
      "&OBJ_DATA=YES"\
      "&REF_SYSTEM=ICRF"\
      "&REF_PLANE=FRAME"\
      "&VEC_TABLE=2"
    return url

  def readData(url, save):
    read_data = requests.get(url)
    src = read_data.text.split('\n')

    data_start_idx = 0
    data_end_idx = 0
    mass_checkonce = True
    for idx, line in enumerate(src):
      if "$$SOE" in line: # start of data
        data_start_idx = idx+1
      if "$$EOE" in line: # end of data
        data_end_idx = idx

    columns = src[data_start_idx-3].split(',')
    heading = [heading.strip() for heading in columns]

    df = pd.DataFrame(data.split(",") for data in src[data_start_idx:data_end_idx])
    df.columns = heading

    # drop unecessary info
    state_vectors = df.drop(['JDTDB','Calendar Date (TDB)',df.columns[-1]], axis=1)
    state_vectors=state_vectors.apply(pd.to_numeric)

    # change km to m
    for i in {'X','Y','Z','VX','VY','VZ'}:
      state_vectors[i] = state_vectors[i]*(1000)

    return state_vectors

  # actually run
  url = URL(target_body, center_body, timestep, start, stop)
  state_vectors = readData(url, save)

  time_index = [dt*x for x in state_vectors.index.tolist()]
  state_vectors.index = time_index

  df_length = (len(state_vectors)-1)*dt

  if save == True:
```

```
    state_vectors.to_csv(f'{path_name}groundtruthdata.csv')

  return state_vectors, time_index, df_length
```

Euler

```
def Gf(df,t):

  p = df.loc[t,:]

  r = (p[0]**2 + p[1]**2 + p[2]**2)**(1/2)
  F = (multiply)*(1/(r**2))

  df.loc[t,'r'], df.loc[t,'F']=r,F

def GAccel(df,t):

  p = df.loc[t,:]

  scale = p[7]/M_p
  norm = 1/p[6]
  T = scale*norm

  ax = -p[0]*T
  ay = -p[1]*T
  az = -p[2]*T

  df.loc[t,'AX'], df.loc[t,'AY'], df.loc[t,'AZ'] = ax, ay, az

def SimEuler(df,dt, df_length, save, t):

  def Next_State(df,t,dt):

    i_t = df.loc[t,:]

    x = i_t[0]+dt*i_t[3]+(1/2)*i_t[8]*(dt**2)
    y = i_t[1]+dt*i_t[4]+(1/2)*i_t[9]*(dt**2)
    z = i_t[2]+dt*i_t[5]+(1/2)*i_t[10]*(dt**2)

    vx = i_t[3]+dt*i_t[8]
    vy = i_t[4]+dt*i_t[9]
    vz = i_t[5]+dt*i_t[10]

    T=t+dt

    df.loc[T,'X'], df.loc[T,'Y'], df.loc[T,'Z'], df.loc[T,'VX'], df.loc[T,'VY'], df.loc[T,'VZ'] = x,y,z,vx,vy,vz

  def Iterate(df,dt,t):
    t = t
    while t+dt <= df_length:

      Gf(df,t)
      GAccel(df,t)
      Next_State(df, t, dt)

      t+= dt

    return t

  final_t = Iterate(df,dt,t)

  Gf(df,final_t)

  if save == True:
    df.to_csv(f'{path_name}Eulerdata.csv')
```

RK4

```
def aT(a,b,*kwargs):
  return tuple([sum(x) for x in zip(a,b,*kwargs)])

def mT(S,a):
  return tuple([S*x for x in a])

def SimRK4(df,dt, df_length, save, t):

  def RK4Accel(pos,M):
    (x,y,z) = pos
    RK4R = (x**2+y**2+z**2)**(1/2)
    RK4F = (multiply)*(1/(RK4R**2))
    RK4Scale = RK4F/M
    RK4Norm = (1/RK4R)
    RK4T = RK4Scale*RK4Norm

    Rawr = (RK4AX, RK4AY, RK4AY) = mT(-RK4T, pos)

    return Rawr

  def RK4_var(df, t, dt):

    p = df.loc[t,:]

    pos = (p[0], p[1], p[2])
    vel = (p[3], p[4], p[5])
    accel = (p[8], p[9], p[10])

    # k_1
    K1S = vel # velocity at t
    K1V = accel # acceleration at t

    #k_2
    K2S = aT(vel,mT((dt/2),K1V)) # velocity at t+h/2
    K2V = RK4Accel(aT(pos,mT((dt/2),K1S)),M_p) # acceleration at t+h/2

    #k_3
    K3S = aT(vel,mT((dt/2),K2V)) # velocity at t+h/2
    K3V = RK4Accel(aT(pos,mT((dt/2),K2S)),M_p) # acceleration at t+h/2

    #k_4
    K4S = aT(vel,mT(dt,K3V))
    K4V = RK4Accel(aT(pos,mT(dt,K3S)),M_p)

    return (K1S, K1V, K2S, K2V, K3S, K3V, K4S, K4V)

  def RK4(df, t, dt, KVARS):

    p = df.loc[t,:]

    pos = (p[0], p[1], p[2])
    vel = (p[3], p[4], p[5])
    accel = (p[8], p[9], p[10])

    (K1S, K1V, K2S, K2V, K3S, K3V, K4S, K4V)  = KVARS

    #S
    S_t = aT(pos, mT((dt/6),aT(K1S,mT(2,K2S),mT(2,K3S),K4S)))

    #V
    V_t = aT(vel, mT((dt/6),aT(K1V,mT(2,K2V),mT(2,K3V),K4V)))

    T = t+dt

    df.loc[T,'X'], df.loc[T,'Y'], df.loc[T,'Z'], df.loc[T,'VX'], df.loc[T,'VY'], df.loc[T,'VZ'] = S_t + V_t

  def Iterate(df,dt,t):
    while t + dt <= df_length:

      Gf(df,t)
      GAccel(df,t)

```

```
    KVARS = RK4_var(df, t, dt)

    RK4(df, t, dt, KVARS)

    t+=dt
  return t

final_t = Iterate(df,dt,t)

Gf(df,final_t)

if save == True:
  df.to_csv(f'{path_name}RK4data.csv')

```

Plotting and Error

matplotlib general code

```
def create_legend(label_list):
  label_list = label_list
  color_list = ['#6CB36B','#33CCCC', '#CC99FF', '#FF5050', '#E86F6F', '#F79E63']
  legend_list = [mpatches.Patch(color=color_list[i], label=f'{label_list[i]}') for i in range(len(label_list))]

  return legend_list, color_list
```

Magnitude Visualization

```
def Mag(df1,df2,df3, index_list, save):

  A = calcR(df1, 'p')
  E = df2['r']
  R = df3['r']
  AV = calcR(df1, 'v')
  EV = calcR(df2, 'v')
  RV = calcR(df3, 'v')

  label_list = ['Actual', 'Euler', 'RK4']
  legend_list, color_list = create_legend(label_list)

  fig, axs = plt.subplots(1, 2, figsize=(20,10))

  #axs = np.array(axs)

  #for ax in axs.reshape(-1):
  axs[0].set_xlabel('time elapsed (s)')
  axs[1].set_xlabel('time elapsed (s)')
  axs[0].set_ylabel('position magnitude (m)')
  axs[1].set_ylabel('velocity magnitude (m/s)')

  axs[0].plot(index_list, A, linestyle='-', linewidth=1, c=color_list[0])
  axs[0].plot(index_list, E, linestyle='-', linewidth=1, c=color_list[1])
  axs[0].plot(index_list, R, linestyle='-', linewidth=1, c=color_list[2])
  axs[1].plot(index_list, AV, linestyle='-', linewidth=1, c=color_list[0])
  axs[1].plot(index_list, EV, linestyle='-', linewidth=1, c=color_list[1])
  axs[1].plot(index_list, RV, linestyle='-', linewidth=1, c=color_list[2])

  #plt.xlim(left=0)
  axs[0].set_ylim(bottom=0)
  axs[1].set_ylim(bottom=0)

  #plt.legend(handles=legend_list, borderaxespad=0.5)

  if save == True:
    plt.savefig(f'{path_name}/MagPosVel.png')

  plt.show()
```

SWE

```
def SWE(R_values, index_list, save):

  SWE = pd.DataFrame()
  SWE['EulerPos'] = (R_values['PAER']/R_values['POR'])**2
  SWE['RK4Pos'] = (R_values['PARR']/R_values['POR'])**2
  SWE['EulerVel'] = (R_values['VAER']/R_values['VOR'])**2
  SWE['RK4Vel'] = (R_values['VARR']/R_values['VOR'])**2

  EP, RP, EV, RV = list(SWE['EulerPos']), list(SWE['RK4Pos']), list(SWE['EulerVel']), list(SWE['RK4Vel'])

  label_list = ['EP', 'RP', 'RV', 'EV']
  legend_list, color_list = create_legend(label_list)

  fig, axs = plt.subplots(3, 2, figsize=(20,30))

  axs = np.array(axs)

  for ax in axs.reshape(-1):
    ax.set_xlabel('time elapsed (s)')
    ax.set_ylabel('error magnitude (1 scale)')

  title_PV = ['Position', 'Velocity']
  title_ERC = ['Euler', 'RK4', 'Combined']

  for j in range(2):
    posvel = title_PV[j]
    for i in range(3):
      ERC = title_ERC[i]
      axs[i,j].title.set_text(f'{posvel}-{ERC}')

  axs[0,0].plot(index_list, EP, linestyle='-', linewidth=1, c=color_list[0])
  axs[1,0].plot(index_list, RP, linestyle='-', linewidth=1, c=color_list[1])
  axs[2,0].plot(index_list, EP, linestyle='-', linewidth=1, c=color_list[0])
  axs[2,0].plot(index_list, RP, linestyle='-', linewidth=1, c=color_list[1])

  axs[0,1].plot(index_list, EV, linestyle='-', linewidth=1, c=color_list[2])
  axs[1,1].plot(index_list, RV, linestyle='-', linewidth=1, c=color_list[3])
  axs[2,1].plot(index_list, EV, linestyle='-', linewidth=1, c=color_list[2])
  axs[2,1].plot(index_list, RV, linestyle='-', linewidth=1, c=color_list[3])

  plt.legend(handles=legend_list, bbox_to_anchor=(1, 2), loc=2, borderaxespad=0.5)

  if save == True:
    SWE.to_csv(f'{path_name}SWE.csv')
    plt.savefig(f'{path_name}SWEPlot.png')

  plt.show()
```

3D Visualization

```
def dynamicplot3D(AE, AR, O, save, ms, acc_ground_truth, k1 = None, k2 = None):
  if acc_ground_truth == True:
    # pregenerated
    O = pd.read_csv('/content/drive/MyDrive/mathIA/PLUTODATA_accurate.csv')
    O.drop(columns = O.columns[0], axis = 1, inplace=True)
    if (k1 and k2) != None:
      AE3D = AE[k1:k2].copy()
      AR3D = AR[k1:k2].copy()
      O3D = O[int(k1*(dt/3600)):int(k2*(dt/3600))].copy()
    else:
      AE3D = AE.copy()
      AR3D = AR.copy()
      #O3D = O.copy()
      O3D = O[0:len(AE)*int((dt/3600))].copy()
  else:
    if (k1 and k2) != None:
```

```
    AE3D = AE[k1:k2].copy()
    AR3D = AR[k1:k2].copy()
    O3D = O[k1:k2].copy()
  else:
    AE3D = AE.copy()
    AR3D = AR.copy()
    O3D = O.copy()

AE3D['colour'] = 'Euler'
AR3D['colour'] = 'RK4'
O3D['colour'] = 'Actual'

df = pd.concat([AE3D, AR3D, O3D])

fig = px.line_3d(df, x='X', y='Y', z='Z', color='colour')
fig.update_traces(marker_size = ms)
fig.show()

if save == True:
  if (k1 and k2) != None:
    fig.write_html(f'{path_name}{k1}:{k2}line3D.html')
  else:
    fig.write_html(f'{path_name}line3D.html')
```

Simulation

```
state_vector, time_index, df_length = extract(
  target_body = target_body,
  center_body = center_body,
  timestep = timestep,
  start = start,
  stop = stop,
  dt = dt,
  save = save_all
)
```

Running Iterations

```
euler_df = state_vector[0:1].copy()
SimEuler(
  df = euler_df,
  dt = dt,
  df_length = df_length,
  save=save_all,
  t=0
)
```

```
RK4_df = state_vector[0:1].copy()
SimRK4(
  df = RK4_df,
  dt = dt,
  df_length = df_length,
  save=save_all,
  t=0
)
```

Error

```
euler_df = pd.read_csv(f'{path_name}finalEulerdata.csv')
euler_df.drop(columns = euler_df.columns[0], axis = 1, inplace = True)
euler_df.index = (np.array(euler_df.index.values.tolist())*dt)
```

```
RK4_df = pd.read_csv(f'{path_name}finalRK4data.csv')
RK4_df.drop(columns = RK4_df.columns[0], axis = 1, inplace = True)
RK4_df.index = (np.array(RK4_df.index.values.tolist())*dt)
```

Difference Error (Component)

```
diffEuler = diff_comp(state_vector, euler_df)
diffRK4 = diff_comp(state_vector, RK4_df)
```

```
AbsDifference(
    diffEuler = diffEuler,
    diffRK4 = diffRK4,
    index_list = time_index,
    save = save_all
)
```

Difference Error

```
R_values = Magnitude(
    df1 = state_vector,
    O_AE = diffEuler,
    O_AR = diffRK4,
    save = save_all
)
```

```
Mag(
    df1 = state_vector,
    df2 = euler_df,
    df3 = RK4_df,
    index_list = time_index,
    save = save_all
)
```

Square Weighted Error

```
SWE(
    R_values = R_values,
    index_list = time_index,
    save = save_all
)
```

3D plot

```
len(state_vector)
```

```
72
```

```
dynamicplot3D(
    AE = euler_df,
    AR = RK4_df,
    O = state_vector,
    save = save_all,
    ms = 1,
    acc_ground_truth = True
)
```

3D plot (section)

```
 dynamicplot3D(
     AE = euler_df,
     AR = RK4_df,
```

```
    O = state_vector,
    save = save_all,
    ms = 1,
    acc_ground_truth = True,
    k1 = 0,
    k2 = 25
)
```

comparing between different errors

```
df = pd.read_csv('/content/drive/MyDrive/mathIA/1_1sec/SWE.csv')
```

```
a = df['EulerPos'].sum()
print(a)
```

```
7.650134581394359e-11
```

```
b = df['RK4Pos'].sum()
print(b)
```

```
7.644638073727034e-11
```

```
a-b
```

```
5.496507667325077e-14
```

Magnitude

```
duration = '500_24hour'
dt=3600*24
```

```
#state_vector = pd.read_csv('/content/drive/MyDrive/mathIA/500_240hour/SWE.csv')
R_values = pd.read_csv(f'/content/drive/MyDrive/mathIA/{duration}/R_values.csv')

df1 = pd.read_csv(f'/content/drive/MyDrive/mathIA/{duration}/groundtruthdata.csv')
df2 = pd.read_csv(f'/content/drive/MyDrive/mathIA/{duration}/Eulerdata.csv')
df3 = pd.read_csv(f'/content/drive/MyDrive/mathIA/{duration}/RK4data.csv')
```

```
index_list = time_index = [dt*x for x in df1.index.tolist()]
```

```
Mag(
    df1 = df1,
    df2 = df2,
    df3 = df3,
    index_list = index_list,
    save = True,
    name = duration
)
```

```
df1['r'] = R_values['POR']
RO_AE, RO_AR = Angle(df1, df2, df3)
```

```
    EM, RM, EA, RA = list(R_values['PAER']), list(R_values['PARR']), RO_AE, RO_AR

    label_list = ['EP', 'RP', 'RV', 'EV']
    legend_list, color_list = create_legend(label_list)

    fig, axs = plt.subplots(3, 2, figsize=(20,30))

    axs = np.array(axs)

    for ax in axs.reshape(-1):
      ax.set_xlabel('time elapsed (s)')
```

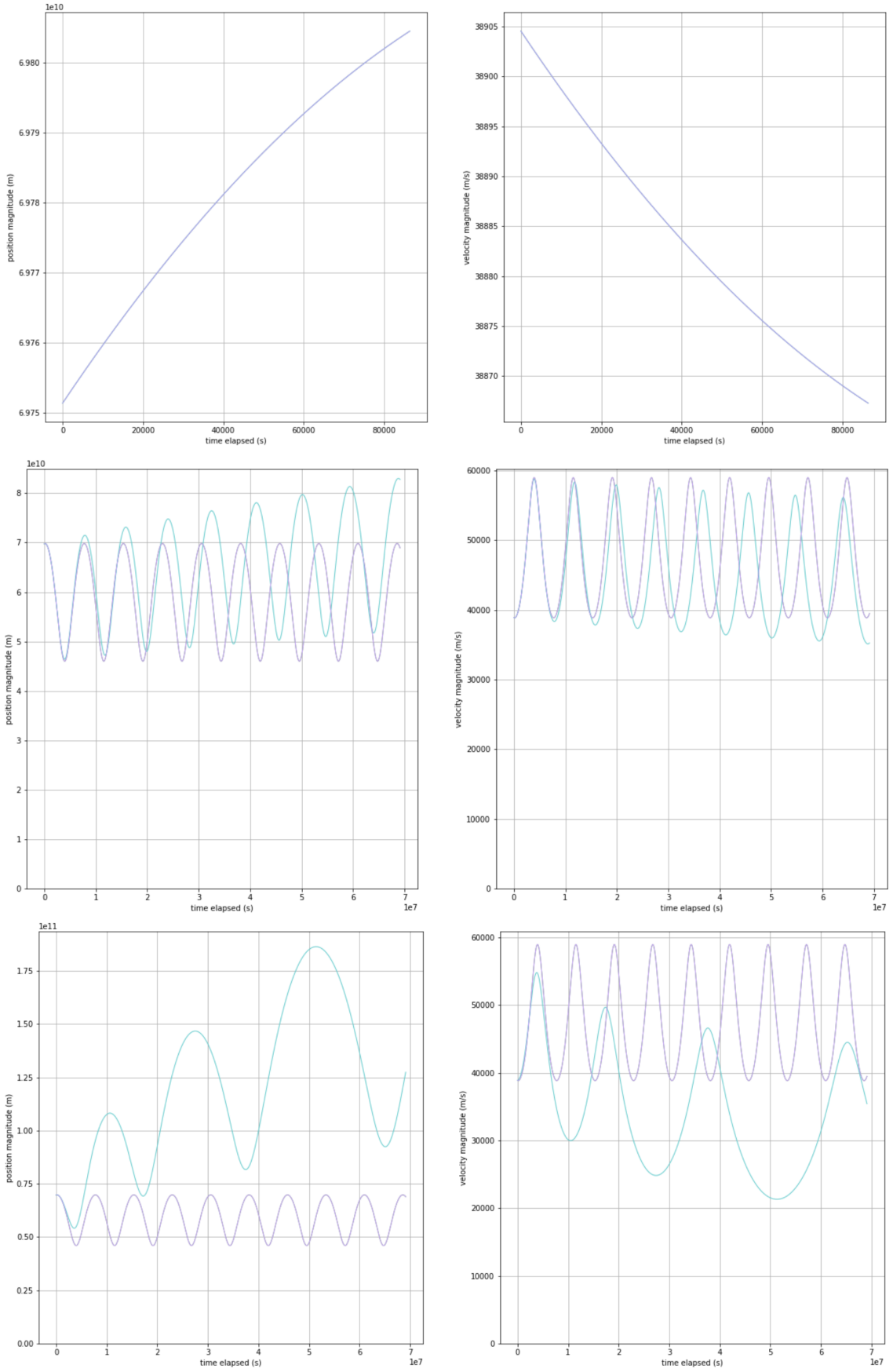
```
    title_PV = ['Magnitude', 'Angle']
    title_ERC = ['Euler', 'RK4', 'Combined']
    y_label = ['absolute difference (m)','angle difference ($radian$)']

    for j in range(2):
      posvel = title_PV[j]
      y = y_label[j]
      for i in range(3):
        ERC = title_ERC[i]
        axs[i,j].title.set_text(f'{posvel}-{ERC}')
        axs[i,j].set_ylabel(f'{y}')

    axs[0,0].plot(index_list, EM, linestyle='-', linewidth=1, c=color_list[0])
    axs[1,0].plot(index_list, RM, linestyle='-', linewidth=1, c=color_list[1])
    axs[2,0].plot(index_list, EM, linestyle='-', linewidth=1, c=color_list[0])
    axs[2,0].plot(index_list, RM, linestyle='-', linewidth=1, c=color_list[1])

    axs[0,1].plot(index_list, EA, linestyle='-', linewidth=1, c=color_list[2])
    axs[1,1].plot(index_list, RA, linestyle='-', linewidth=1, c=color_list[3])
    axs[2,1].plot(index_list, EA, linestyle='-', linewidth=1, c=color_list[2])
    axs[2,1].plot(index_list, RA, linestyle='-', linewidth=1, c=color_list[3])

    plt.legend(handles=legend_list, bbox_to_anchor=(1, 2), loc=2, borderaxespad=0.5)

    plt.savefig(f'/content/drive/MyDrive/mathIA/{duration}/2MagAngPlot.png')

    plt.show()
```

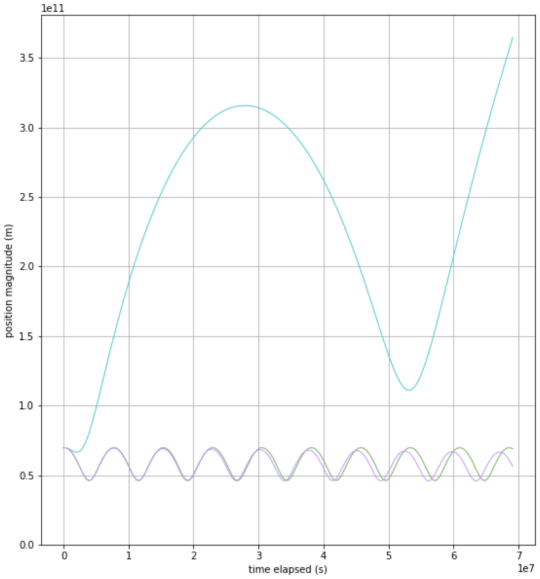
```
dynamicplot3D(
    AE = df2,
    AR = df3,
    O = df1,
    save = False,
    ms = 1,
    acc_ground_truth = True
)
```

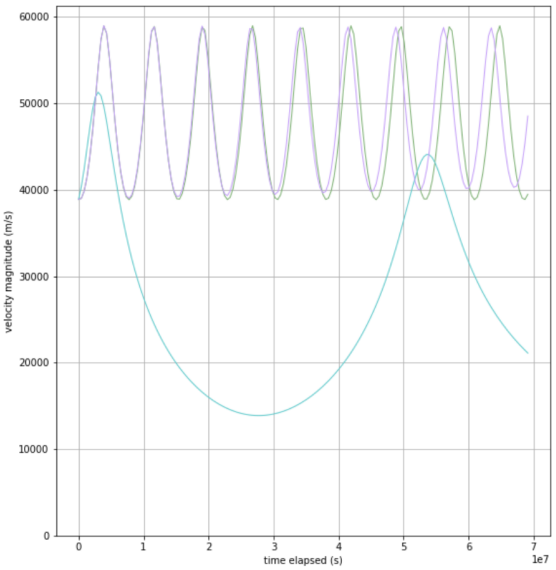
```
MagAng(
    df1 = df1,
    df2 = df2,
    df3 = df3,
    R_values = R_values,
    index_list = time_index,
    save = save_all
)
```

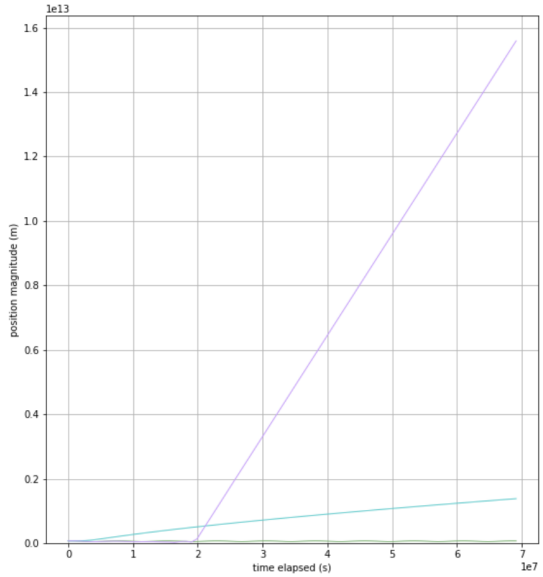
6.3 Full Simulation Results

Figure 17: Magnitude Plots of Ground Truth (green), Euler (blue), RK4 (purple) data for $h = 1sec, 1hour, 1day, 5days, 10days$ respectively




1e11
position magnitude (m)
time elapsed (s)
1e7


velocity magnitude (m/s)
time elapsed (s)
1e7


1e13
position magnitude (m)
time elapsed (s)
1e7

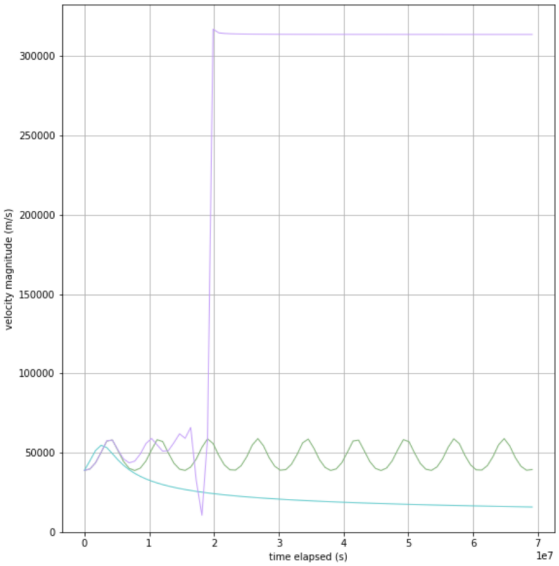
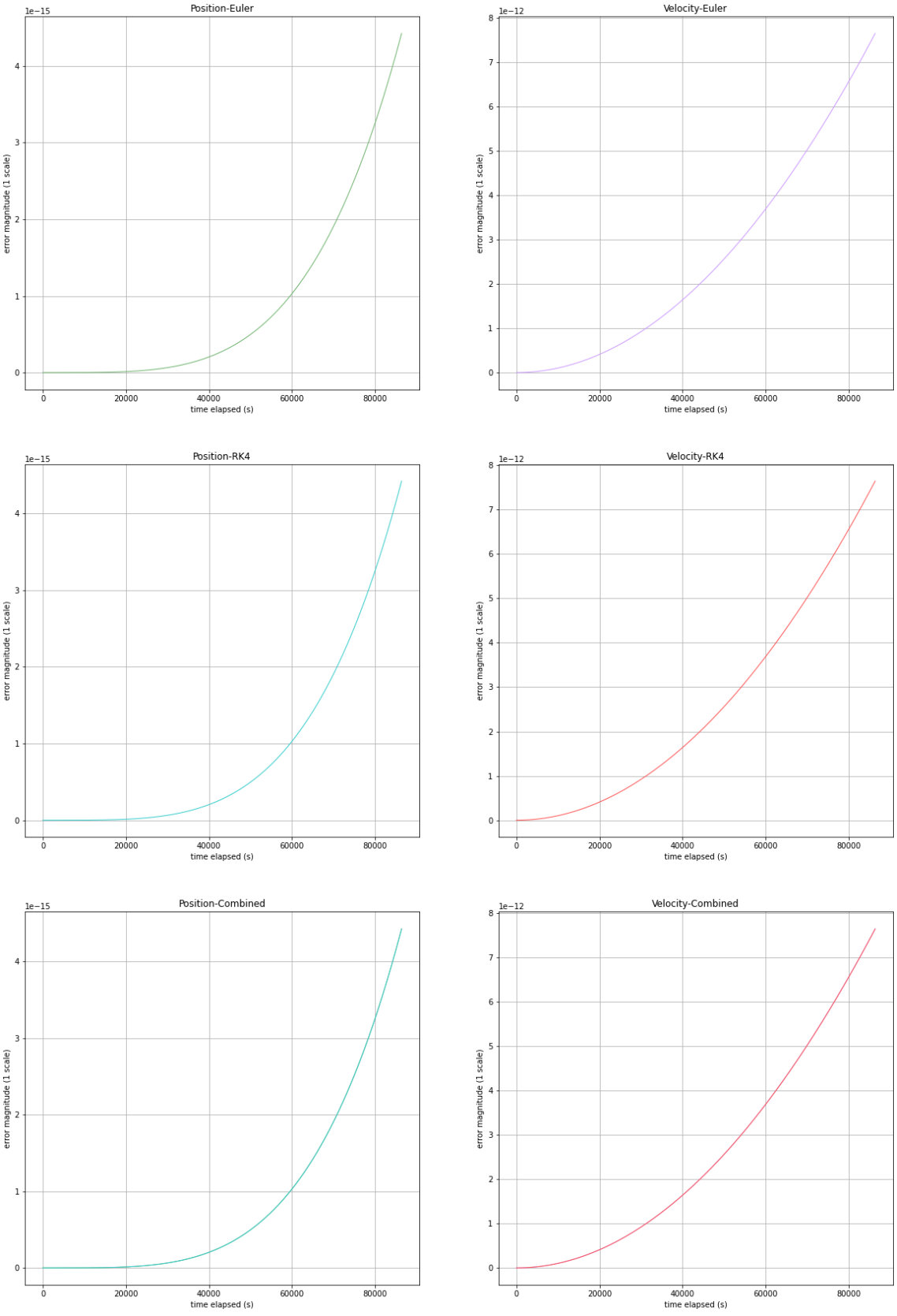
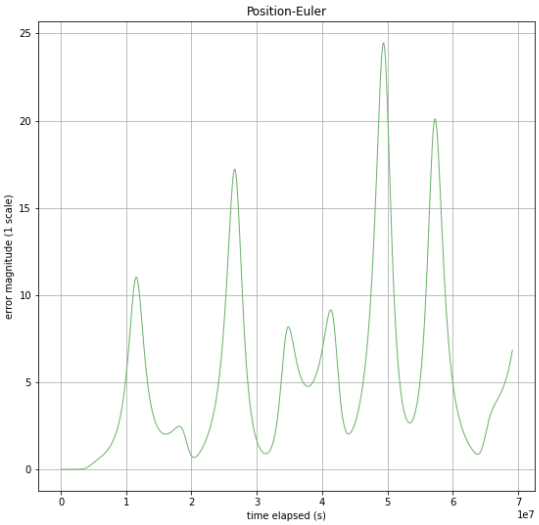
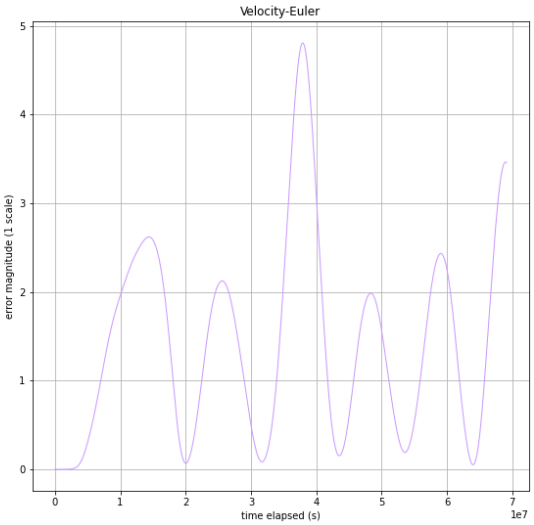
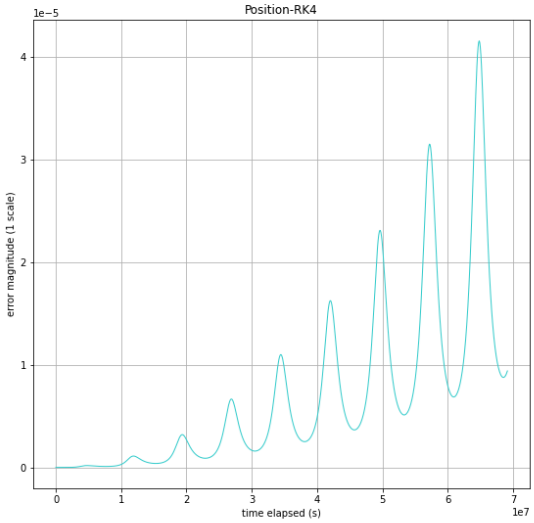

velocity magnitude (m/s)
time elapsed (s)
1e7

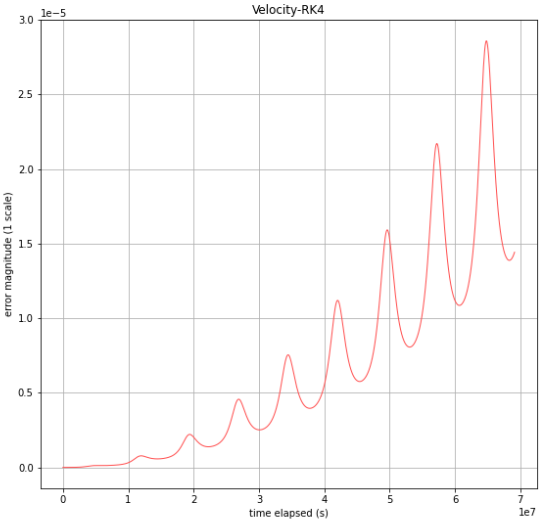
Figure 18: SWE Plots of Ground Truth (green), Euler (blue), RK4 (purple) data for $h = 1sec, 1hour, 1day, 5days, 10days$ respectively (one page each h value)

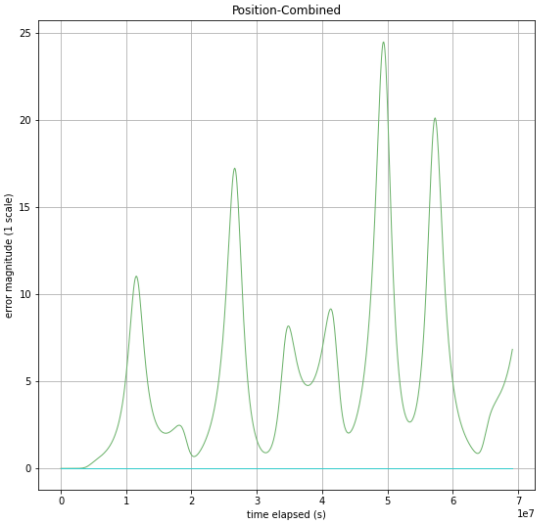


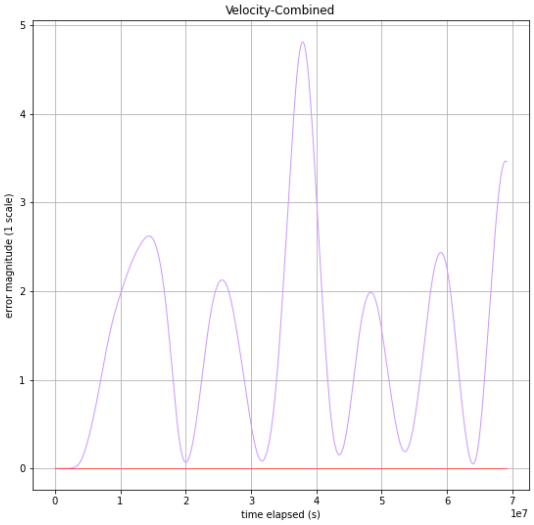

Position-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

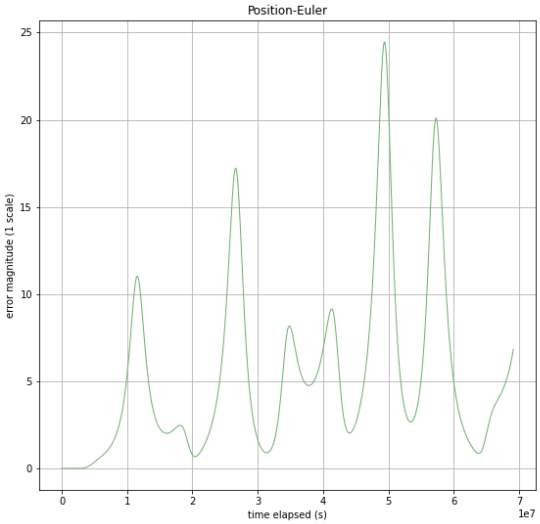

Velocity-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

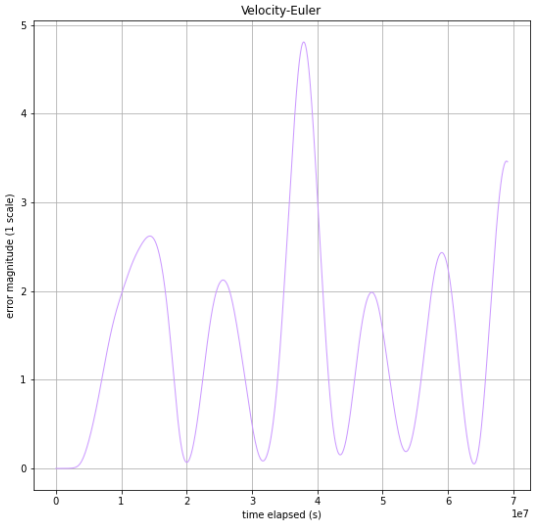

Position-RK4
1e−5
error magnitude (1 scale)
time elapsed (s)
1e7

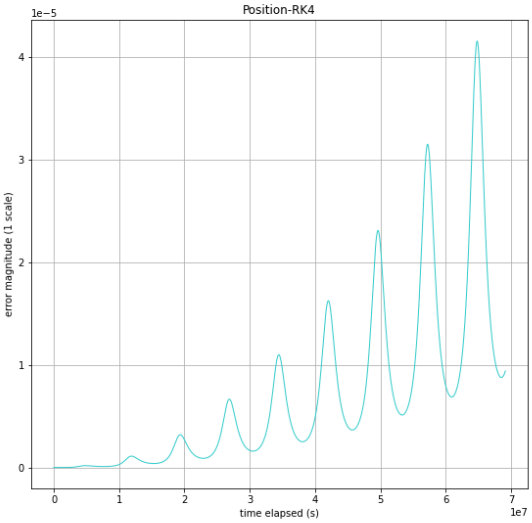

Velocity-RK4
1e−5
error magnitude (1 scale)
time elapsed (s)
1e7

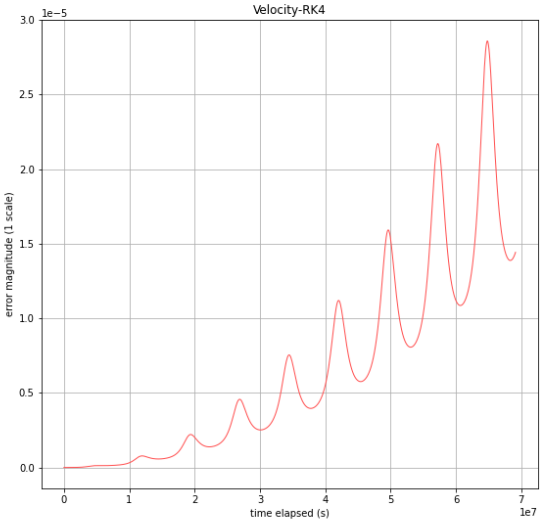

Position-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

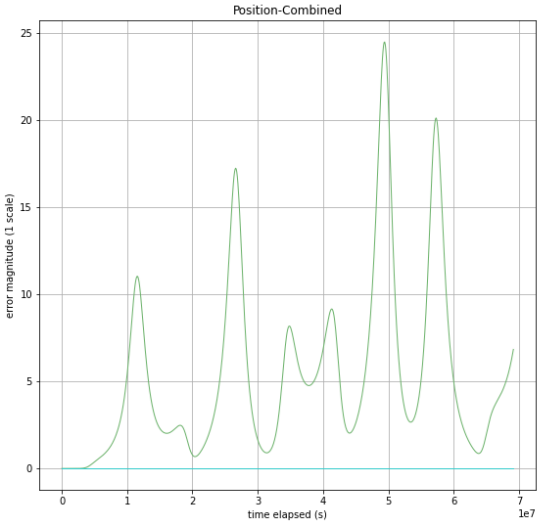

Velocity-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

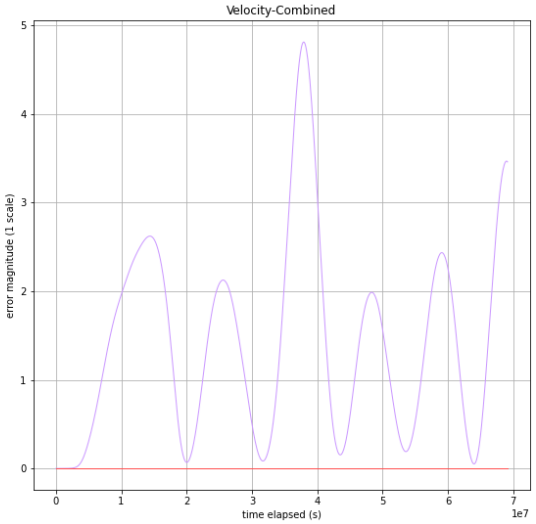

Position-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

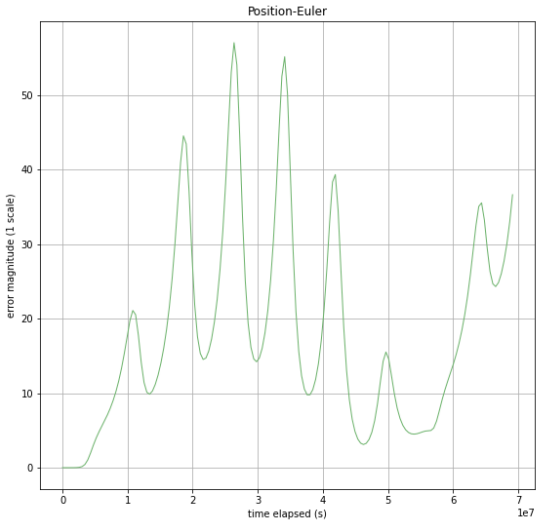

Velocity-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

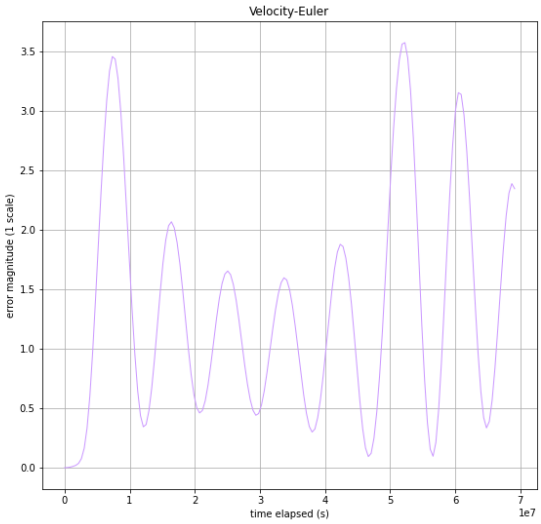

Position-RK4
1e−5
error magnitude (1 scale)
time elapsed (s)
1e7

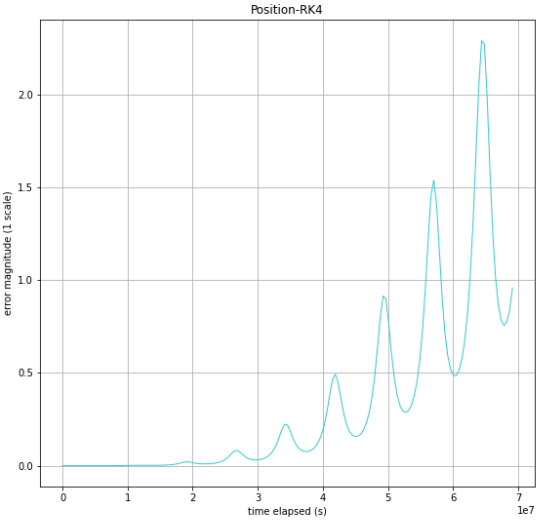

Velocity-RK4
1e−5
error magnitude (1 scale)
time elapsed (s)
1e7

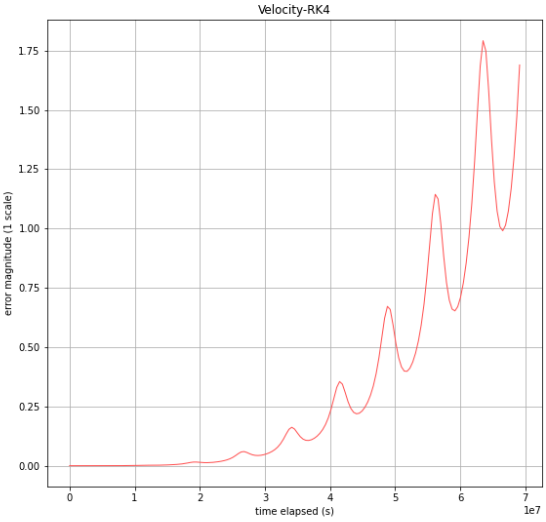

Position-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

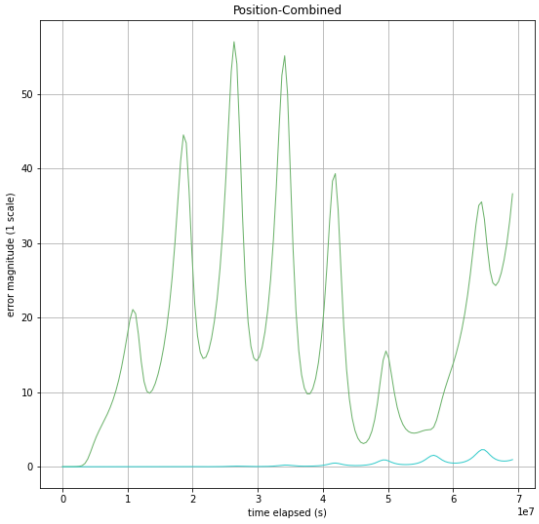

Velocity-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

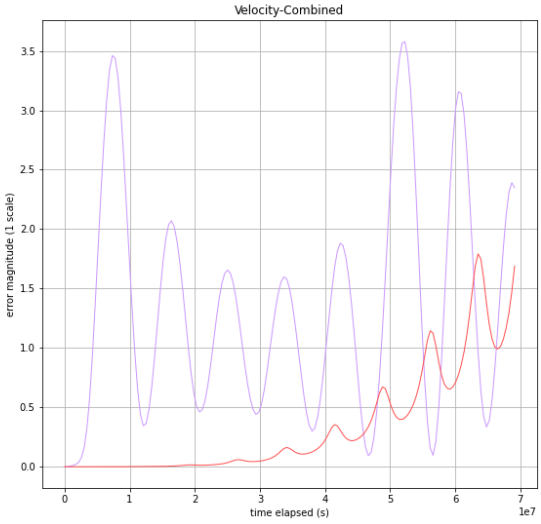

Position-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

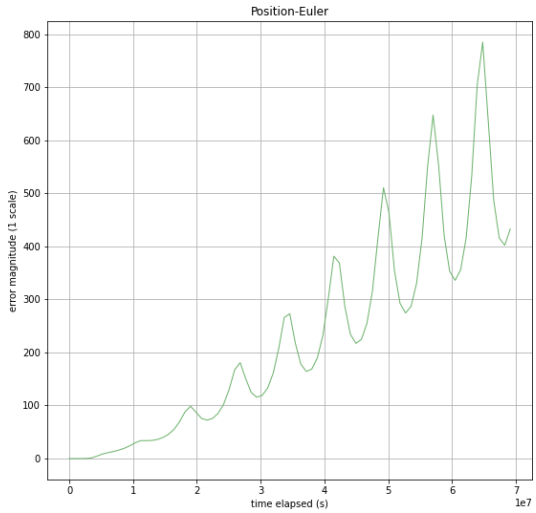

Velocity-Euler
error magnitude (1 scale)
time elapsed (s)
1e7

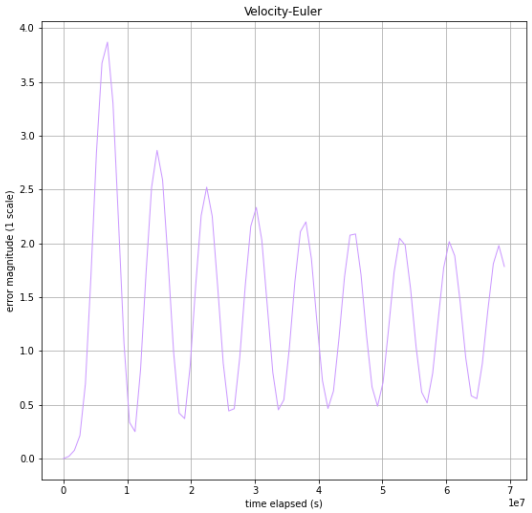

Position-RK4
error magnitude (1 scale)
time elapsed (s)
1e7

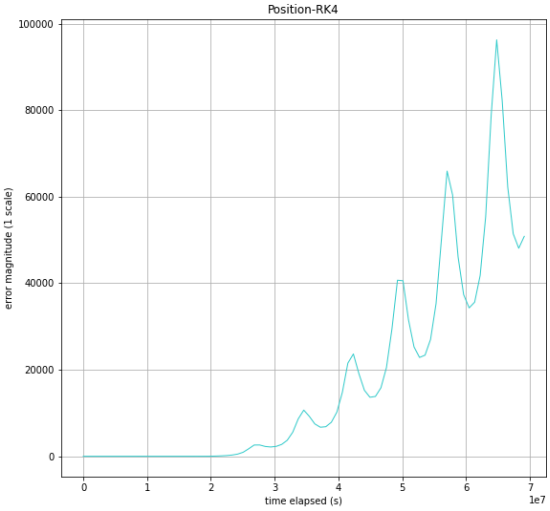

Velocity-RK4
error magnitude (1 scale)
time elapsed (s)
1e7

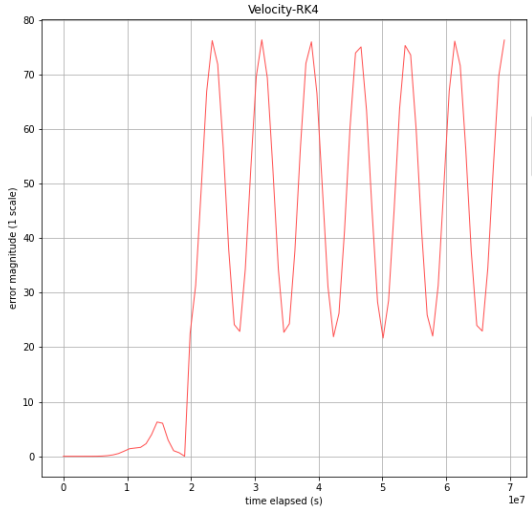

Position-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

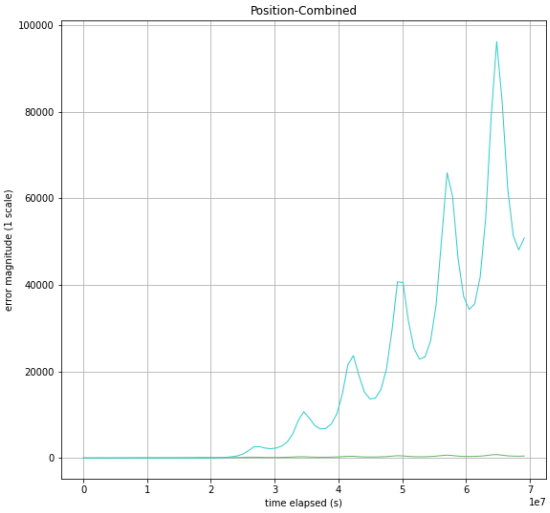

Velocity-Combined
error magnitude (1 scale)
time elapsed (s)
1e7


Position-Euler
error magnitude (1 scale)
time elapsed (s)
1e7


Velocity-Euler
error magnitude (1 scale)
time elapsed (s)
1e7


Position-RK4
error magnitude (1 scale)
time elapsed (s)
1e7


Velocity-RK4
error magnitude (1 scale)
time elapsed (s)
1e7


Position-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

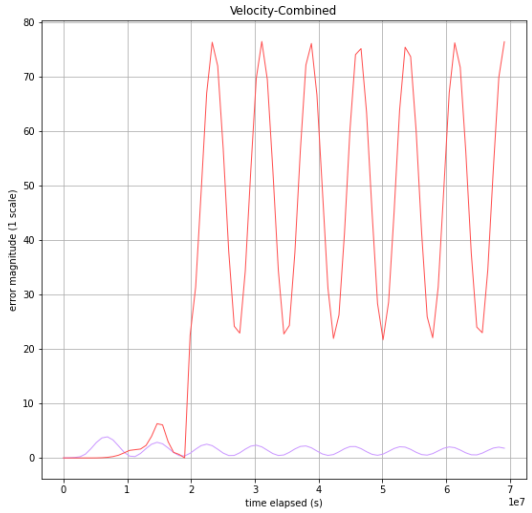
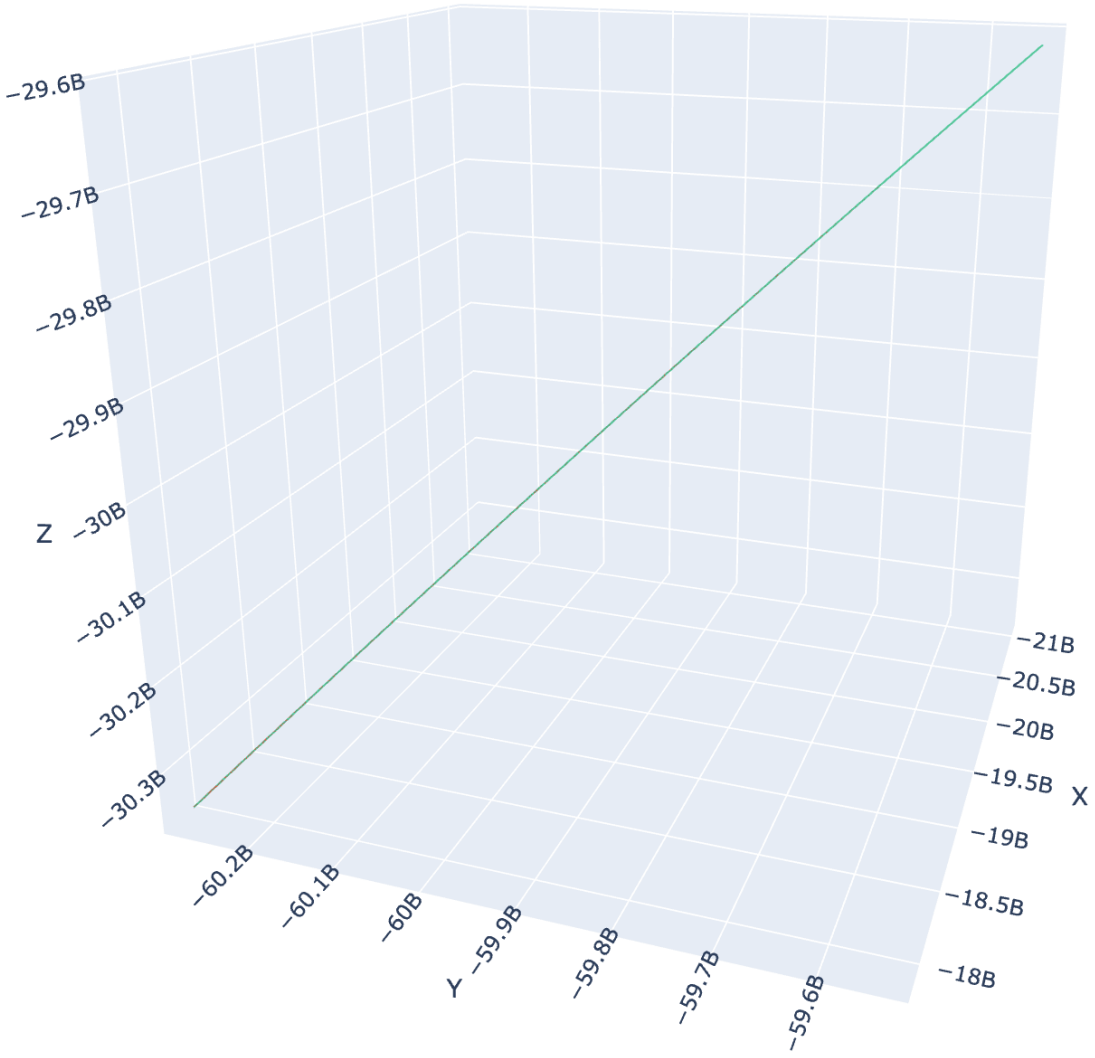
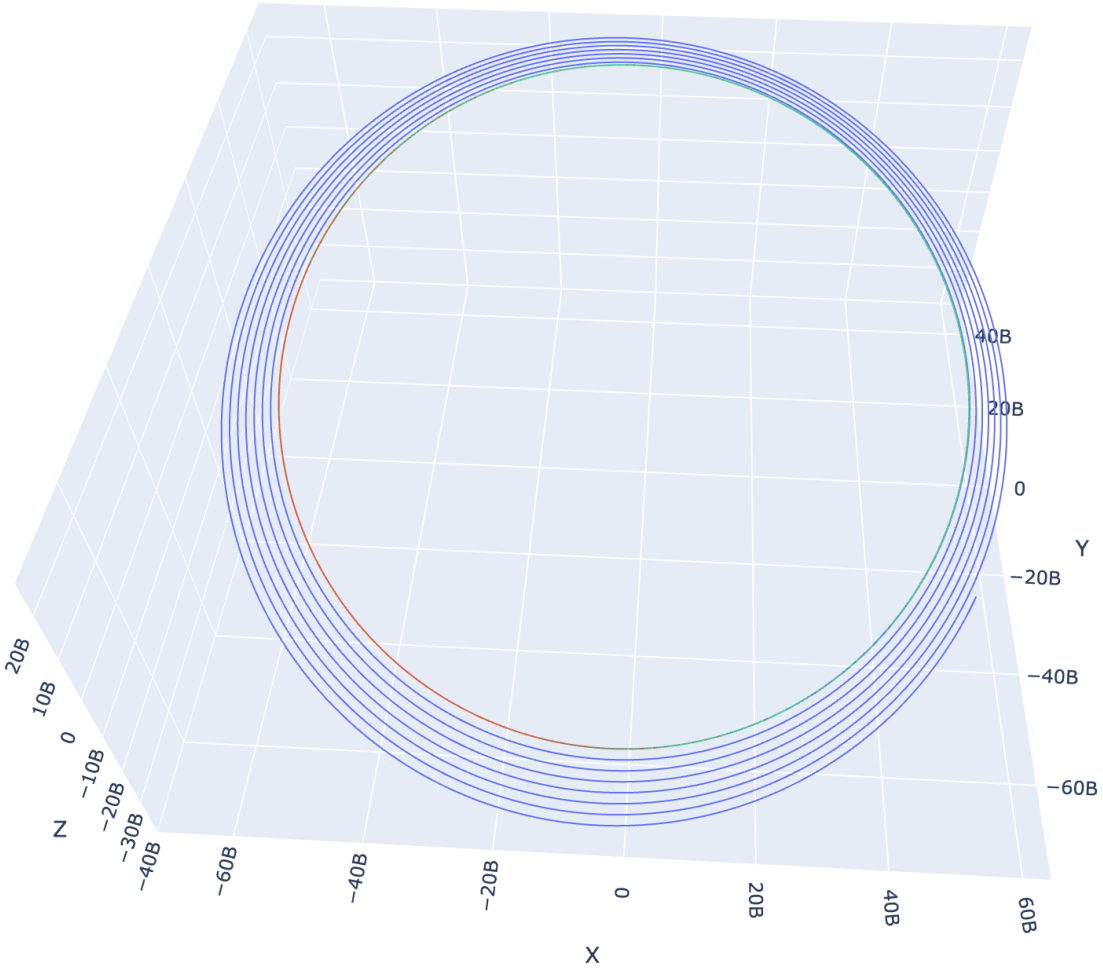
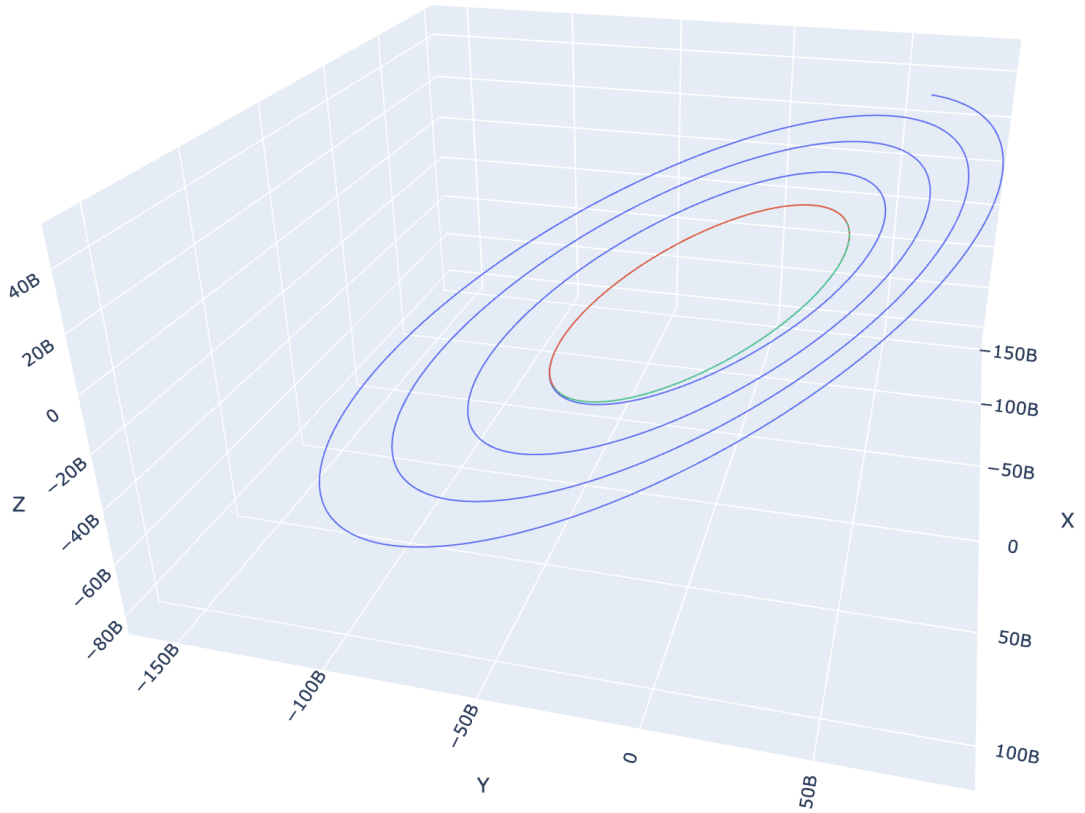
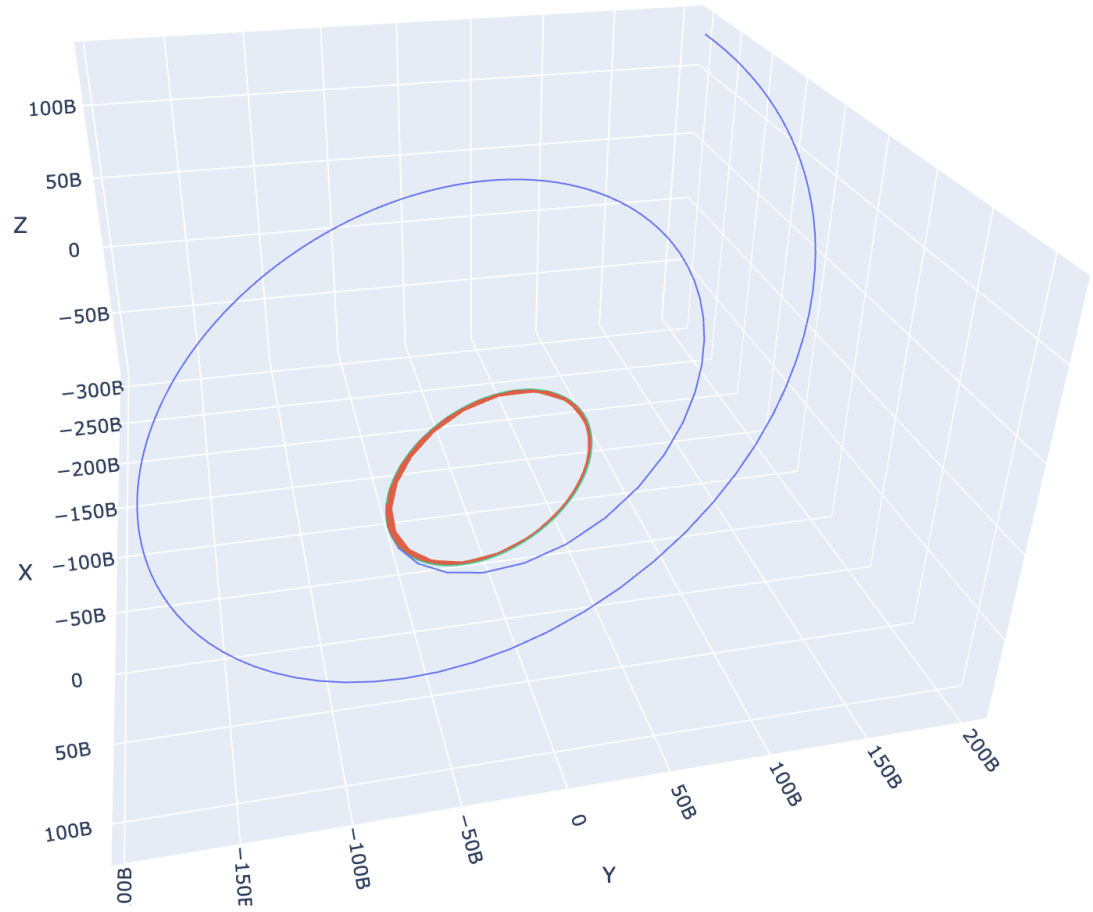

Velocity-Combined
error magnitude (1 scale)
time elapsed (s)
1e7

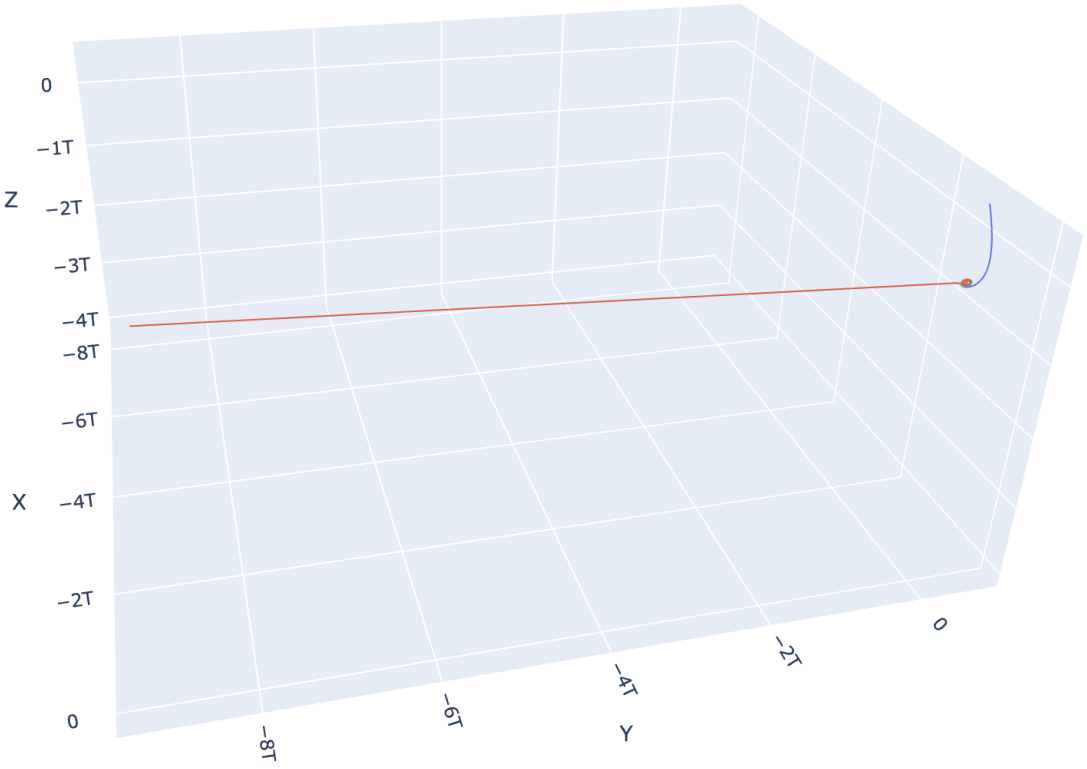
Figure 19: 3D Visualization of Ground Truth (green), Euler (blue), RK4 (red) data for $h = 1sec, 1hour, 1day, 5days, 10days$ respectively




40B
20B
0
Y
−20B
−40B
−60B
20B
10B
0
−10B
−20B
−30B
−40B
Z
−60B
−40B
−20B
0
20B
40B
60B
X


40B
20B
0
−20B
−40B
−60B
−80B
Z
−150B
−100B
−50B
0
50B
Y
−150B
−100B
−50B
0
50B
100B
X


100B
50B
Z
0
−50B
−300B
−250B
−200B
−150B
X
−100B
−50B
0
50B
100B
−150B
−100B
−50B
0
50B
100B
150B
200B
Y


0
−1T
Z
−2T
−3T
−4T
−8T
−6T
X
−4T
−2T
0
−8T
−6T
−4T
−2T
0
Y