AE4878 - Mission Geometry and Orbit Design
Part 3 - Integrator Evaluation (Euler vs RK4)

Ali Nawaz
3ME, LR, Delft University of Technology.

September 2, 2018

1 Introduction

The goal of this assignment is to first test Euler and Runge Kutta 4 integrator schemes on a basic kinematic problem. Later the same integration schemes are applied to an orbital mechanics problem of a Geostationary Earth Orbit (GEO) satellite.

But before we start, why do we need integrator schemes? In real life, majority of the engineering problems are discrete time problems. What does it mean? It means that the sensors, which extract valuable information of a system or it's environment, sample observations at some intervals/steps. Sensor engineering says a big "no" to differentiate the observed data samples. It magnifies the noise in the dataset. To avoid this problem observations are recorded via means such that the required information can be found by integrating the observations. Integration of observed samples is the "to go method" to gain further insight into the dynamics of the system.

Section 1.1 outlines the co-ordinate system used through out this assignment along with the constant parameters. Section 2 focuses on the application of integrator schemes on a kinematic particle problem. Finally Section 3 focuses on the application of integrator schemes to analyse the dynamics of a GEO satellite.

1.1 Co-ordinate System and Parameters Values

Coordinate systems used throughout this assignment is illustrated with the aid of Figure 1 and 2 [1]pg.261. TLE elements are obtained in the Kepler elements format. However, to process them further to gain insight about the satellite dynamics transformation to Cartesian co-ordinate is necessary.

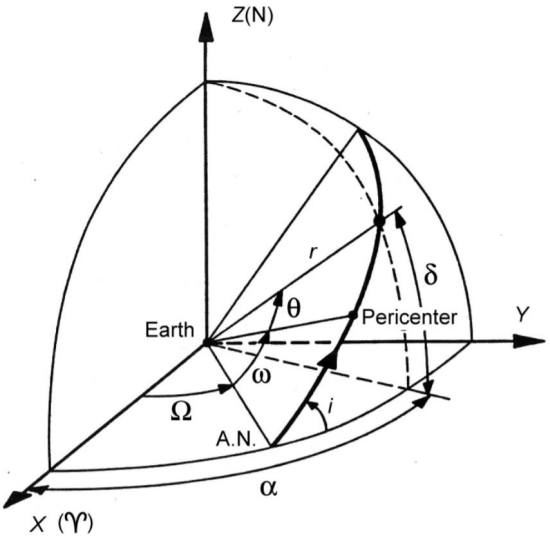

Figure 1: Three dimensional Kepler coordinate frame.

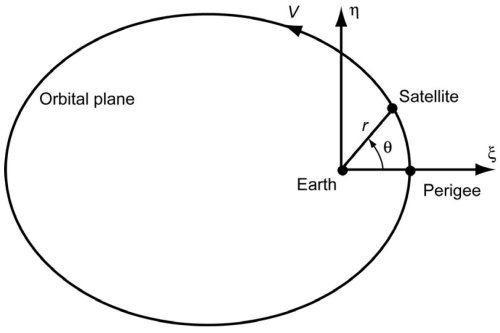

Figure 2: Coordination for positional transformation from Kepler to Cartesian

Some of the standard parameters used through out the assignment in outlined in Table 3.

Table 1: Standard parameters used for the assignment.

Parameter	Definition	Value	Unit	Source
μ	Standard gravitational parameter of Earth	$3.986004419 \cdot 10^{14}$	m^3/s^2	en.wikipedia.org/wiki/Standard_gravitational_parameter
Solar day	Solar rotation period of the Earth	86400	s	https://en.wikipedia.org/wiki/Day
R_E	Radius of Earth	$6378.136 \cdot 10^3$	m	AE4878 Mission Geometry and Orbit Design R. Noomen, TU Delft Lecture - Intro,Slide 16, OCDM parameter list
h_{GEO}	Height of GEO satellites	$35786 \cdot 10^3$	m	en.wikipedia.org/wiki/Geostationary_orbit

2 INTEG - 1

A racing car is considered as a particle and kinematics equations of motion is used to estimate the state of its dynamics as it progresses in time. The car is assumed to have an initial velocity of 0 m/s and a constant acceleration of $2m/s^2$. Drag and other forms of resistance are ignored. First an analytical solution is estimated. Later numerical solutions are obtained with the aid of Euler and Runge Kutta integrators. The integrators are tested and verified on a simple problem before application on the actual kinematics. To conclude this section, accuracy analysis and trade off between the integrator techniques is conducted.

2.1 Analytic problem

Since no form of resistant forces act on the car and it moves with constant acceleration. The car can be simplified in a particle problem of kinematics. Analytic expressions for distance travelled and velocity of a particle is outlined with the aid of Equation 1. A Matlab script is presented in 4.1.

Distance travelled, x

$$x = x_0 + \dot{x}_0(t_f - t_0) + \frac{1}{2}\ddot{x}(t_f - t_0)^2$$

Where x is position, $\dot{x}$ is velocity, $\ddot{x}$ is acceleration and t_f and t_0 are the final and initial times. (1)

End velocity $\dot{x}$ is given by:

$$\dot{x} = \dot{x}_0 + \ddot{x}(t_f - t_0)$$

While the end values of the distance travelled and the final velocity are outlined in Equation 2.

Distance travelled, x

$$x = 0 + 0 \cdot (60 - 0) + \frac{1}{2} \cdot 2 \cdot (60 - 0)^2 = 3600m$$

Where x is position, $\dot{x}$ is velocity, $\ddot{x}$ is acceleration and t_f and t_0 are the final and initial times. (2)

End velocity $\dot{x}$ is given by:

$$\dot{x} = 0 + 2(60 - 0) = 120 \text{ m/s}$$

2.2 Euler integration

First an Euler integration scheme is tested out with a simple scalar test program. Once the program is verified, the logic is carried forward to establish the kinematics with an Euler integrator. The Euler integrator will integrate the motion of the car (position and velocity) over the time interval [0s,60s]. General expression of Euler integration is presented with the aid of Equation 3 [2][Lecture - Integrator v4-10, Slide - 14].

$$x(t_0 + h) \approx x_0 + h\phi = \eta(t_0 + h)$$

$$\phi_{Euler} = \dot{x}_0 = f(t_0, x_0)$$

Where x is the physical truth value and h is the step size. (3)

η is the numerical approximation and ϕ is the increment function.

2.2.1 Scalar test program

A scalar test program is written to verify the algorithm logic. The program is presented in Script 4.1. The idea is to replicate the processes outlined in [2][Lecture - Integrator v4-10, Slide 16]. The function is e^{2t} and is run from time t = 1s to t =2s. The results obtained via the test program and the ones presented in the lecture slides is outlined in Table 2. Based on the tabulated values, the algorithm logic outlined in Section 4.1 is reliable.

Table 2: Verification scalar test program

Parameters	Lecture Slides	Estimated
x(1)	7.389	7.389056099
dx/dt(1)	14.778	14.77811220
x(2)	22.167	22.16716830

2.2.2 Kinematics with Euler

Now that the Euler integrator logic is tested out on a sample scalar problem, it can be extended to the kinematic problem. The Kinematic Euler integrator is outlined in 4.1. Fundamental expressions of the Euler scheme presented in the script is outlined in Equation 4.

$$\bar{x} = \begin{bmatrix} x(t_0+h) \\ \dot{x}(t_0+h) \end{bmatrix} \approx \begin{bmatrix} x(t_0) \\ \dot{x}(t_0) \end{bmatrix} + h \cdot \phi = \eta(t_0+h)$$

$$\phi_{Euler} = \dot{\bar{x}}_0 = f(t_0, x_0)$$

Thus the expression can be simplified to :

$$\begin{bmatrix} x_{i+1} \\ \dot{x}(i+1) \end{bmatrix} \approx \begin{bmatrix} x(i) \\ \dot{x}(i) \end{bmatrix} + h \cdot \phi = \eta(t_0+h) \tag{4}$$

Where, ϕ:

$$\phi = \begin{bmatrix} \dot{x}(i) \\ \ddot{x}(i) \end{bmatrix} = \begin{bmatrix} \dot{x}(i) \\ 2 \end{bmatrix}$$

Note $2m/s^2$ is the constant acceleration.

Table 3 outlines the results obtained via the Euler integration scheme, compared against analytic results. For every order decrease in step size, the % difference decreases by an order while the no. of iterations (proportional to computational effort) increases by an order too. Choice of accuracy depends on the purpose of application. If 2% difference is a reasonable deviation for a certain application than step size of 1 meets the criteria. But if the maximum expected deviation is 0.02% then a step size of 0.01 needs to be chosen. % Difference is estimated using Equation 5. Furthermore time is also a major factor, for offline operations time might not be a real factor but for online operations time could easily become the driving factor. From the presented values, best output time is 0.031s and the worst is 2.69s.

$$\%\text{Difference} = \frac{|\text{Table Values - Estimated Values}|}{\text{Table Values}} \cdot 100\% \tag{5}$$

Table 3: Trade off between step size and quality of results for Euler.

Step size	Distance Traveled [m]	Analytical Distance [m]	% Difference	End Velocity [m/s]	Analytical Velocity [m/s]	% Difference	No. of iterations	Run time [s]
10	3000.00	3600	16.67	120	120	0	6	0.031
1	3540.00	3600	1.667	120	120	0	60	0.031
0.1	3594.00	3600	0.1667	120	120	0	600	0.033
0.01	3599.40	3600	0.01667	120	120	0	6000	0.069
0.001	3599.94	3600	0.001667	120	120	0	60000	2.69

Irrespective of the time step chosen the end velocity perfectly matches the analytic velocity. The reason being that acceleration is constant. Velocity is a linear function of time for constant acceleration. Euler scheme results in a linear relation between time and velocity as outlined in Equation 4.

2.3 Runge Kutta 4 integrator

Euler integrator was verified and the results of Euler kinematic scheme were in line with the analytic scheme.The test phase for RK4 is skipped. If the RK4 implementation is correct, then the results will be $\approx$ or equal to the analytic solution. RK4 integrator will integrate the motion of the car (position and velocity) over the time interval [0s,60s]. General expression of RK4 integrator is presented with the aid of equations in Figure 3 [2][Lecture - Integrator v4-10, Slide - 19]. Scripts are presented in Section 4.1 for further reference.

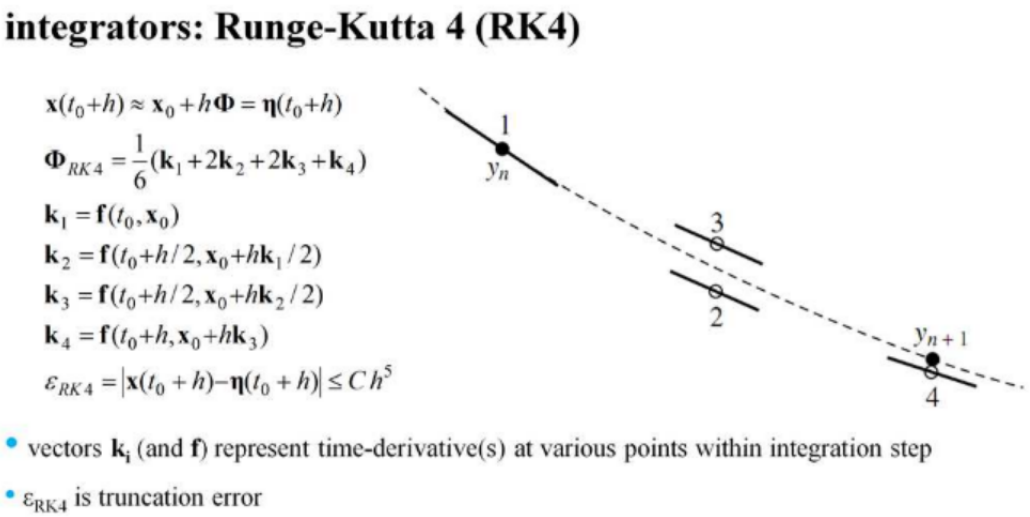

Figure 3: Runge Kutta 4 integrator problem formulation.

2.3.1 Kinematics with RK4

The state vector is chosen the same as before [x $\dot{x}$] i.e. [position velocity]. The parameter for the increment function ϕ are now updated with the aid of parameters $k_1 - k_4$ as shown previously in Figure 3. Update of k parameters and consecutively ϕ requires the update of initial state, x_0. This is shown with the aid of the following algorithm:

```
k1 = [ x0(2,1) x_ddot]'; % [ velocity from initial vector [m/s], constant acceleration ...
    [m/s^2] ]

x0_k2 = x0 + (dt/2).*k1; % Update the initial state vector to half of a time step
k2 = [ x0_k2(2,1) x_ddot]'; % Find k2 using this information

x0_k3 = x0 + (dt/2).*k2; % Update initial state using the information of k2
k3 = [ x0_k3(2,1) x_ddot]'; % Find k3

x0_k4 = x0 + dt.*k3; % Update initial state using information of k3
k4 = [ x0_k4(2,1) x_ddot]'; % Find k4

phi = (1/6).* (k1 + 2.*k2 + 2.*k3 + k4); % Define incremental function

x = x0 + dt.*phi; % State update with RK4 integration
```

Table 4 outlines the results obtained for kinematics with the aid of RK4

Table 4: Trade off between step size and quality of results for RK4.

Step size	Distance Traveled [m]	Analytical Distance [m]	% Difference	End Velocity [m/s]	Analytical Velocity [m/s]	% Difference	No. of iterations	Run time [s]
30	3600	3600	0	120	120	0	2	0.032
20	3600	3600	0	120	120	0	3	0.032
10	3600	3600	0	120	120	0	6	0.033
1	3600	3600	0	120	120	0	60	0.034
0.1	3600	3600	0	120	120	0	600	0.035
0.01	3600	3600	0	120	120	0	6000	0.083
0.001	3600	3600	0	120	120	0	60000	2.783

Irrespective of the time step chosen the end velocity perfectly matches the analytic velocity. The reason being that acceleration is constant and ofcourse RK4 works like magic.

2.4 Accuracy analysis and trade off

For Euler integrator the number of derivative evaluations per iteration iteration step is 1. This can be clearly observed from the increment function of Euler in Equation 3. While for Runge Kutta 4 integrator the number of derivative evaluations per iteration is 4. This was outlined in terms of equations with the aid of Figure 3. Table 5 outlines the step size, no. of iterations, no. of derivative evaluations, run time and the percentage difference between numerical and analytic results.

Table 5: Accuracy analysis and trade off.

Toy Car Problem		*Euler Integrator Scheme*				*Runge Kutta 4 Integrator Scheme*			
Step Size	**No. of iterations**	**% Difference Distance [m]**	**% Accuracy**	**No. of derivative evaluations**	**Run Time [s]**	**% Difference Distance [m]**	**% Accuracy**	**No. of derivative evaluations**	**Run Time [s]**
30	2	50.00	50.000	2	0.030	0	100.00	8	0.032
20	3	33.33	66.6667	3	0.030	0	100.00	12	0.032
10	6	16.67	83.3333	6	0.031	0	100.00	24	0.033
1	60	1.667	98.3333	60	0.031	0	100.00	240	0.034
0.1	600	0.1667	99.8333	600	0.033	0	100.00	2400	0.035
0.01	6000	0.01667	99.9833	6000	0.069	0	100.00	24000	0.083
0.001	60000	0.001667	99.9983	60000	2.69	0	100.00	240000	2.783

Figure 4 illustrates the accuracy of the end distance as a function of the number of derivative evaluations. For the first four data points (steps: 30, 20, 10 and 1) RK4 integrator clearly gives the best performance. However, from 5th data point onward (steps: 0.1, 0.01, 0.001 ..) Euler integrator provides same accuracy as RK4 integrator but does so with lower number of derivative evaluations! If accuracy is the primary objective and computational costs the secondary, then one should go for RK4 integrator for the first four step sizes and for step size lower than or equal to 0.1 one should switch to Euler integrator. If computational effort(no. of derivation evaluations) is the primary objective, then Euler is the first choice. Since Euler requires a quarter of of derivative evaluations of Runge Kutta 4 integrator.

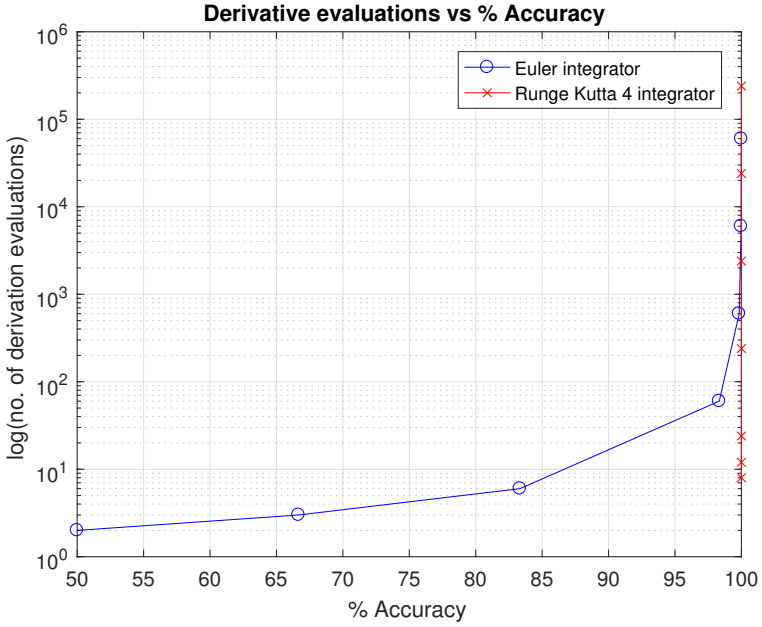

Figure 4: Number of derivation evaluations against % accuracy. Number of derivation evaluations is plotted in log scale to see the features.

And last but not least a cost function is chosen which is a combination of both accuracy and computational effort. The cost function is outlined with the aid of Equation 6. Higher the accuracy and lower the computational effort,

higher the value of cost function. Equal weights are put on both the accuracy and the computational effort. For Euler integrator the cost functions are [25.0000, 22.2222, 13.8889 , 1.6389 , 0.1664 , 0.0167, 0.0017]' for steps outlined in Table 5. Similarly for RK4 integrator the cost functions are [12.5000, 8.3333, 4.1667, 0.4167, 0.0417, 0.0042, 0.0004]'. Clearly for equal weights Euler is the leading choice.

$$\text{Cost Function} = \frac{\text{\% Accuracy}}{\text{No. of derivative evaluations}} \tag{6}$$

To conclude, RK4 is chosen when accuracy is the primary requirement. Euler is chosen is computational effort is the primary requirement. If equal weights are put on accuracy and no. of derivation evaluations, Euler is the leading choice. Script for this trade off and accuracy analysis is presented in Section 4.1

3 INTEG - 3

The aim of this chapter is to extend the scope of integrator to the application of orbital mechanics. Euler and Runge Kutta 4 integrator outlined in Section 2 are reused here. With the exception of a different state vector. However, the underlying process is similar. A Matlab script is presented for both Euler and Runge Kutta 4 integrators. The script is first tested out to verify the initial and final values with known values presented in Lecture 4[2]. Later they are implemented on the actual problem. To conclude this Section a trade off is conducted between Euler and RK4 integrator for this specific purpose.

3.1 Transformation

To facilitate transformation from Kepler to Cartesian coordinate frame and vice verse, transformation script written as part of Assignment 1 is used. The scripts are presented in Section 4.2. To provide an overview on the accuracy of transformation from Kepler to Cartesian and vice verse, verification tables from Assignment 1 is outlined here. Tables 6 and 7 outline the respective transformation accuracies. Judging the offsets from the tables, it can be concluded that the algorithm outputs are reasonable for implementation in the integrator.

Table 6: Cartesian input elements and their corresponding Kepler output elements. For verification purposes values from the lecture slides is presented. The output values are presented upto 10 significant figures.

Input Element	Input Value	Unit	Output Element	Output Value	Verification Value	% Offset	Unit
x	-2700816.14	[m]	a	6.787746876e+06	6787746.891	2.261024896e-07	m
y	-3314092.80	[m]	e	7.311020662e-04	0.000731104	2.645074976e-04	-
z	5266346.42	[m]	i	51.68714486	51.68714486	8.318441965e-10	deg
V_x	5168.606550	[m/s]	Ω	127.5486706	127.5486706	1.954917023e-08	deg
V_y	-5597.546618	[m/s]	ω	74.21979912	74.21987137	9.734967710e-05	deg
V_z	-868.878445	[m/s]	θ	24.10034902	24.10027677	2.998057252e-04	deg
			E	24.08324991	24.08317766	3.000200981e-04	deg
			M	24.06615651	24.06608426	0.07132715978	deg

Table 7: Kepler input elements and their corresponding Cartesian output elements. For verification purposes values from the lecture slides is presented. The output values are presented upto 10 significant figures.

Input Element	Input Value	Unit	Output Element	Output Value	Verification Value	% Offset	Unit
a	6787746.891	m	x	-2.700816139e+06	-2700816.14	2.092896011e-08	[m]
e	0.000731104	-	y	-3.314092801e+06	-3314092.80	3.075290532e-08	[m]
i	51.68714486	deg	z	5.266346421e+06	5266346.42	1.288021402e-08	[m]
Ω	127.5486706	deg	V_x	5.168606557e+03	5168.606550	5.861478138e-06	[m/s]
ω	74.21987137	deg	V_y	-5.597546622e+03	-5597.546618	4.906786338e-06	[m/s]
M	24.06608426	deg	V_z	-8.688784455e+02	-868.878445	2.075441659e-06	[m/s]
			E	24.08317766	24.08317766	8.476854218e-09	deg
			θ	24.10027676	24.10027677	2.241870733e-08	deg

3.2 Euler Integration

The fundamentals outlined in Equation 3 are true for this Section as well. With the exception of the change is state vectors. The state vector and the corresponding Euler upgrade is outlined with the aid of Equation

7.

The state vector is given by:

$$\bar{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

The increment function $\phi = \dot{\bar{x}}$ can be expressed as:

$$\phi = \dot{\bar{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}$$

The acceleration terms can be derived from the potential function of a Kepler two body problem. (7)

Potential function V(r) $= -\frac{\mu}{r}$, where r $= \sqrt{x^2 + y^2 + z^2}$ and $\mu = \mu_{Earth}$

$$\ddot{x} = \frac{\delta V}{\delta x} = \frac{\delta \dot{x}}{\delta t} = \frac{-\mu x}{r^3}$$
$$\ddot{y} = \frac{\delta V}{\delta y} = \frac{\delta \dot{y}}{\delta t} = \frac{-\mu y}{r^3}$$
$$\ddot{z} = \frac{\delta V}{\delta z} = \frac{\delta \dot{z}}{\delta t} = \frac{-\mu z}{r^3}$$

Thus, the increment function ϕ simplifies to:

$$\phi = \dot{\bar{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{-\mu x}{r^3} \\ \frac{-\mu y}{r^3} \\ \frac{-\mu z}{r^3} \end{bmatrix}$$

The Euler integrator update step can be now be expressed as outlined in Equation 8.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ \dot{x}_{i+1} \\ \dot{y}_{i+1} \\ \dot{z}_{i+1} \end{bmatrix} \approx \begin{bmatrix} x_i \\ y_i \\ z_i \\ \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{bmatrix} + h \cdot \phi = \eta(t_0 + h)$$

Where, ϕ: (8)

$$\phi = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \\ \frac{-\mu x_i}{r^3} \\ \frac{-\mu y_i}{r^3} \\ \frac{-\mu z_i}{r^3} \end{bmatrix}$$

3.2.1 Test program

A test program is constructed with the aid of Equation 7. Corresponding Matlab script is presented in Section 4.3. Table 8 outlines the estimated result with the Matlab test script against the results presented in Lecture 4, Slides 17,18 [2]. First the Kepler elements [a = 7378.137km, e=0.1, i=Ω=ω=M= 0 degree] are transformed to Cartesian co-ordinates [estimated states at t=0s] before running Euler integrator.

Table 8: Results obtained via Euler test algorithm compared against results outlined in Lecture 4.

State vector Element	Estimated state vector at t = 0s	Analytic state vector at t = 0s	% Difference	Estimated state vector at t = 10s	Analytic state vector at t = 10s	%Difference	Estimated state vector at t = 20s	Analytic state vector at t = 20s	%Difference
x [m]	6640323.300	6640323.30	0.00	6640323.30	6640323.30	0.00	6639419.3190	6639419.32	1.506156059e-08
y [m]	0.00	0.00	0.00	81258.83989	81258.84	1.353698792e-07	162517.6798	162517.68	
z [m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ẋ [m/s]	0.00	0.00	0.00	-90.39810484	-90.40	0.0021	-180.7759080	-180.78	0.0023
ẏ [m/s]	8125.883989	8125.88	4.855256383e-05	8125.883989	8125.88	4.855256383e-05	8124.778019	8124.78	2.438219865e-05
ż [m/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

3.2.2 Orbit propagation with Euler integrator

Judging the results, the Euler test algorithm is perfectly on point. The observed differences are the differences due to different decimal points. All the values are 100% accurate up to the decimal places presented in the lecture slides. The same algorithm can now be extended to Cartesian parameter estimation of the GEO satellite. First coordinate transformation is applied to transform from given Kepler elements [e = 0, i = Ω = ω = M = 0 degrees] to Cartesian coordinates. At the end, transformation is applied from Cartesian to Kepler to access the accuracy of the Cartesian element propagation by Euler integrator. This step is facilitated with the aid of transformation algorithm written as part of Assignment 1. Table 9 outlines the results obtained for varying step sizes against the true Kepler elements a and e. Note that the true semi major axis is the sum of the radius of Earth and the orbital height of GEO satellite. Values are taken from Table 1. Choice of accuracy again depends on the purpose of application. Lower time step results in very high run times. This is illustrated with the aid of Table 12. If the focus is on accuracy, even after time steps of 5s Euler does a pretty bad job. If the primary objective is accuracy, a clear trade off is significant increase in computational time and effort.

Table 9: Accuracy analysis and trade off for end values. End value is 30 solar days.

Step Size [s]	Estimated semi major axis [m]	True semi major axis [m]	% Difference	Estimated Eccentricity	True Eccentricity	% Difference	Total % Difference	No. of Iterations	No. of Deriv. Eval.
500	147975508.4	42164136	250.9511	0.1298	0	12.9812		5184	5184
50	72754663.92	42164136	72.5511	0.01043830616	0	1.0438		51840	51840
5	47319700.94	42164136	12.2274	0.00105541615	0	0.10554		518400	518400

3.3 Runge Kutta 4 Integration

The fundamentals outlined in the expression in Figure 3 are true for this Section as well. With the exception of the change is state vectors. The state vector resembles the one outlined in Equation 7. The state vector along with the process of updating k parameters and ϕ is outlined with the aid of Equation 9.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ \dot{x}_{i+1} \\ \dot{y}_{i+1} \\ \dot{z}_{i+1} \end{bmatrix} \approx \begin{bmatrix} x_i \\ y_i \\ z_i \\ \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{bmatrix} + h \cdot \phi = \eta(t_0 + h)$$

Where,$\phi = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

$$k_1 = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \\ \frac{-\mu x_i}{r^3} \\ \frac{-\mu y_i}{r^3} \\ \frac{-\mu z_i}{r^3} \end{bmatrix} ; \; k_2 = \begin{bmatrix} \dot{x}_{ik2} \\ \dot{y}_{ik2} \\ \dot{z}_{ik2} \\ \frac{-\mu x_{ik2}}{r^3} \\ \frac{-\mu y_{ik2}}{r^3} \\ \frac{-\mu z_{ik2}}{r^3} \end{bmatrix} ; \; k_3 = \begin{bmatrix} \dot{x}_{ik3} \\ \dot{y}_{ik3} \\ \dot{z}_{ik3} \\ \frac{-\mu x_{ik3}}{r^3} \\ \frac{-\mu y_{ik3}}{r^3} \\ \frac{-\mu z_{ik3}}{r^3} \end{bmatrix} ; \; k_4 = \begin{bmatrix} \dot{x}_{ik4} \\ \dot{y}_{ik4} \\ \dot{z}_{ik4} \\ \frac{-\mu x_{ik4}}{r^3} \\ \frac{-\mu y_{ik4}}{r^3} \\ \frac{-\mu z_{ik4}}{r^3} \end{bmatrix} \quad (9)$$

Where $\bar{x}_{ikx}$ is updated as:

$$\bar{x}_{ik2} = \bar{x}_i + \frac{h}{2}k_1$$

$$\bar{x}_{ik3} = \bar{x}_i + \frac{h}{2}k_2$$

$$\bar{x}_{ik4} = \bar{x}_i + hk_3$$

3.3.1 Test program

A test program is constructed with the aid of Equation 9. Corresponding Matlab script is presented in Section 4.3. Table 10 outlines the estimated result with Matlab RK4 test script, verified against the values presented in Lecture 4 Slide 21,22 [2].

Table 10: Results obtained via RK4 test algorithm compared against the results outlined in Lecture 4.

State vector Element	Estimated state vector at t = 0s	Analytic state vector at t = 0s	% Difference	Estimated state vector at t = 10s	Analytic state vector at t = 10s	%Difference	Estimated state vector at t = 20s	Analytic state vector at t = 20s	%Difference
x [m]	6640323.30	6640323.30	0.00	6639871.315	6639871.32	7.530266247e-08	6638515.4199	6638515.45	4.534146274e-07
y [m]	0.00	0.00	0.00	81257.00	81257.00	0.00	162502.93	162502.93	0.00
z [m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$\dot{x}$ [m/s]	0.00	0.00	0.00	-90.40	-90.40	0.00	-180.78	-180.78	0.00
$\dot{y}$ [m/s]	8125.88	8125.88	0.00	8125.33	8125.33	0.00	8123.67	8123.67	0.00
$\dot{z}$ [m/s]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Judging the results it can be concluded that the test algorithm outlined in Section 4.3 can be extended to Cartesian parameter estimation with RK4. It is important to note that apart from position x all the other parameters are on point. With x being off in order of magnitude $\approx$ between 10^{-7} and 10^{-8}. This is an acceptable number. It can be concluded that the test algorithm logic is perfect and can be extended for the real application.

3.3.2 Orbit propagation with Runge Kutta 4 integrator

Judging the results of RK4 test algorithm, the same algorithm is now extended to conduct Cartesian parameter estimation of GEO satellite. A Matlab script is outlined in Section 4.3 to facilitate the RK4 integrator. First coordinate transformation is applied to transform from given Kepler elements [e = 0, i = Ω = ω = M = 0 degrees] to Cartesian coordinates. At the end, transformation is applied from Cartesian to Kepler to access the accuracy of the Cartesian element propagation by RK4 integrator. The results of this are outlined in Table 11.

Table 11: Accuracy analysis and trade off for end values. End value is 30 solar days.

Step Size [s]	Estimated semi major axis [m]	True semi major axis [m]	% Difference	Estimated Eccentricity	True Eccentricity	% Difference	No. of Iterations	No. of Deriv. Eval.
500	42164121.748	42164136	3.3825e-05	2.5369e-07	0	2.5369e-05	5184	20736
50	42164135.9998607	42164136	3.3028e-10	2.4828e-12	0	2.4828e-10	51840	207360
5	42164136.000	42164136	9.1356e-12	7.0755e-14	0	7.0755e-12	518400	2073600

For RK4 the answer is straight forward, even with step size as high as 5000 (shown in Table 12), it clearly outperforms the accuracy of Euler. Unlike Euler, higher step size are tolerable with RK4. If the desired accuracy can be achieved with a higher time step why go for all the computational effort that comes with lower time steps?

3.4 Accuracy analysis and trade off

Table 12 outlines the step size, no. of iterations, no. of derivative evaluations, run times and the percentage difference between the numerical and analytic estimations.

Table 12: Accuracy analysis and trade off.

GEO Satellite		*Euler Integrator Scheme*				*Runge Kutta 4 Integrator Scheme*			
Step Size	**No. of iterations**	**% Difference semi major axis [m]**	**% Accuracy**	**No. of derivative evaluations**	**Run Time [s]**	**%Difference semi major axis[m]**	**% Accuracy**	**No. of derivative evaluations**	**Run Time [s]**
5000	518	3348.5272	-3248.8272	518	0.015	0	100.00	2072	0.021
500	5184	250.9511	-150.9511	5184	0.165	0	100.00	20736	0.170
50	51840	72.5511	27.4489	51840	21.500	0	100.00	207360	21.580
5	518400	12.2274	87.7726	518400	2703.533	0	100.00	2073600	2906.858
2.5	1036800	6.4646	93.5354	1036800	-	0	100.00	4147200	14583.714

Figure 5 illustrates the accuracy of the end semi-major axis estimation as a function of the number of derivative evaluations. Each of the stencils in Figure 5 represent the step sizes from Table 12. It can be clearly observed that only once the step size is reduced to 2.5, Euler provides a reasonable accuracy (93.5354% from Table 12) weighing in the computational effort of RK4 for equal time steps. However, RK4 already provides a much better accuracy for higher time steps (e.g. 5000, refer to Table 12). This is both computational effort-wise and accuracy-wise attractive! RK4 is clearly the best choice for this application.

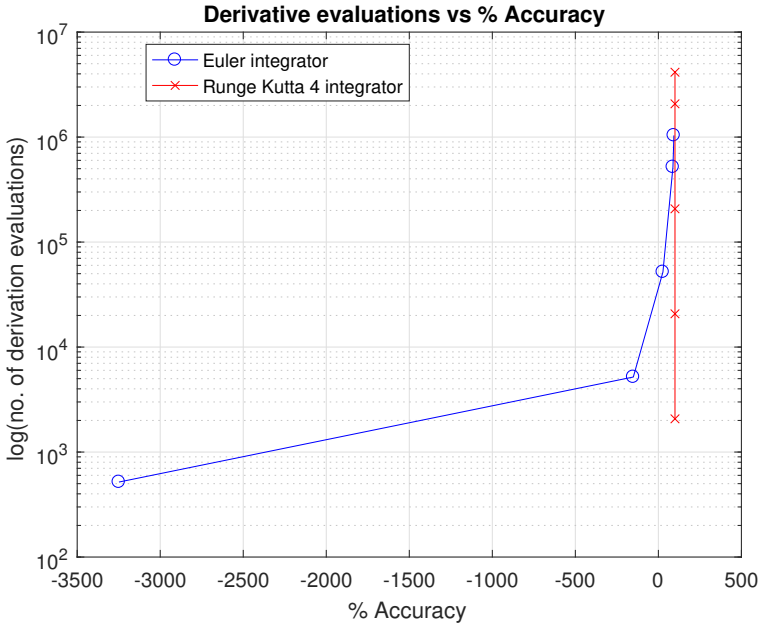

Figure 5: Number of derivation evaluations against % accuracy. Number of derivation evaluations is plotted in log scale to see the features.

4 Matlab Scipt

4.1 Script INTEG 1

Following is the script for analytic expression.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% AE4878 Mission Geometry and Orbit Design %%%%%%%%%%%%%
%%%%% Assignment 3 - Week 4 - Integrators - Version 1 %%%%%
%%%%% Author Info: Ali Nawaz; Student ID - 4276477 %%%%%%%%
%%%%% Assigned tasks : INTEG-1, INTEG - 3 %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% ////////////// INTEG - 1 //////////////

%% ##### 1. Analytical expression for distance and velocity #####
close all; clear all; clc;
% Expression for distance in meters
% s = s0 + u(tf-t0) + 0.5a(tf - t0)^2 => x = x0 + (x_dot0)*(tf- t0) + 0.5*(x_ddot)*(tf -t0)^2

x0 = 0; % initial position [m]
x_dot0 = 0; % initial velocity [m/s]
x_ddot = 2; % constant acceleration [m/s^2]
t0 = 0; % initial time [s]
tf = 60; % final time [s]

% Final distance [m]
x = x0 + (x_dot0)*(tf-t0) + 0.5*(x_ddot)*(tf-t0)^2;

% Expression for final velocity
% v = u + a(tf-t0) => xdot = xdot0 + xddot*(tf-t0)
x_dot = x_dot0 + x_ddot*(tf -t0); % final velocity in [m/s]
```

Following is the script for testing the Euler algorithm logic:

```
%% Euler code logic test with scalar x = exp(2*t), dx/dt = 2*exp(2*t)
test_time_initial = 1; %initial test time [s]
initial_state_test = [ exp(2*test_time_initial)]';
test_time = 2; % total test time [s]
test_dt = 1; % test time step [s]
test_res =[initial_state_test];

for test = test_time_initial+test_dt:test_dt:test_time
    test_phi = 2*exp(2*test_time_initial);
    test_val = initial_state_test + test_dt*test_phi;
end
```

Following is the script for application of Euler integrator on kinematics problem and corresponding analytic solution.

```
%% ##### 2. Euler integration for distance velocity #####
close all; clear all; clc;
% initial state vector : [ distance at t=0 , velocity at t = 0 ]'
x0 = [ 0 0]'; % [ initial distance [m] , intial velocity [m/s] ]
dt = 0.01; % time step [s]
t0 = 0; % initial time [s]
tf = 60; % final time [s]

x_ddot = 2; % Constant acceleration [m/s^2]

res = [x0]; % store results of Euler integration here.

% Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
for t = t0+dt:dt:tf
    t;
    phi = [ x0(2,1) x_ddot]'; % incremental function
    x = x0 + dt.*phi; % Euler integration
    res = [res, x]; % Storing results row1 : pos, row2: vel, col: iteration number + 1
    x0 = x; % Updating initial state for next iteration
end
%% Analytical results for given time series
timeseries = t0+dt:dt:tf ; % Time series used for numerical calculation.
res_ana = [x0, [ 0.5*x_ddot.*timeseries.^2; x_ddot.*timeseries]]; % Analytical results for ...
    the chosen time series
```

Following is the script for solving the kinematics problem with an RK4 integrator:

```
%% ##### 3. Runge Kutta 4 integration for position and velocity #####
%% ACTUAL RUNGE KUTTA INTEGRATOR FOR KINEMATICS PROBLEM
close all; clear all; clc;
% initial state vector
x0 = 0; % initial position [m]
x_dot0 = 0; % initial velocity [m/s]
x_ddot = 2; % constant acceleration [m/s^2]

x0 = [ 0 0]'; % [ initial distance [m] , intial velocity [m/s] ]
dt = 1; % time step [s]
t0 = 0; % initial time [s]
tf = 60; % final time [s]

res = [x0]; % store results of Euler integration here.

% Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
for t = t0+dt:dt:tf
    k1 = [ x0(2,1) x_ddot]'; % [ velocity from initial vector [m/s], constant acceleration ...
        [m/s^2] ]

    x0_k2 = x0 + (dt/2).*k1; % Update the initial state vector to half of a time step
    k2 = [ x0_k2(2,1) x_ddot]';% Find k2 using this information

    x0_k3 = x0 + (dt/2).*k2;% Update initial state using the information of k2
    k3 = [ x0_k3(2,1) x_ddot]';% Find k3

    x0_k4 = x0 + dt.*k3;% Update initial state using information of k3
    k4 = [ x0_k4(2,1) x_ddot]';% Find k4
```

```
28
29      phi = (1/6).* (k1 + 2.*k2 + 2.*k3 + k4); % Define incremental function
30
31      x = x0 + dt.*phi; % RK4 integration
32      res = [res, x]; % Storing results row1 : pos, row2: vel, col: iteration number + 1
33      x0 = x; % Updating initial state for next iteration
34  end
```

Following script is used to provide insight into the accuracies of Euler and Runge Kutta 4 integrator to facilitate trade off.

```
1  %% Exercise 4
2
3  % Accumulated data after multiple runs
4
5  step_sizes = [ 30 20 10 1 0.1 0.01 0.001];
6  deriv_eval_euler = [ 2 3 6 60 600 6000 60000]; % No. of derivative evaluations for Euler
7  deriv_eval_rk4 = 4.*deriv_eval_euler; % No. of derivative evaluations for RK4
8
9  acc_euler = [ 50.00 66.6667 83.3333 98.3333 99.8333 99.9833 99.9983 ]; % Percentage accuracy ...
       Euler
10 acc_rk4  = [ 100 100 100 100 100 100 100]; % Percentage accuracy Runge Kutta 4
11
12 diff_euler = 100.-acc_euler;
13 diff_rk4 = 100.- acc_rk4;
14 figure(1)
15 semilogy(acc_euler, deriv_eval_euler,'b-o', acc_rk4, deriv_eval_rk4,'r-x');
16 title('Derivative evaluations vs % Accuracy');
17 xlabel('% Accuracy');
18 ylabel('log(no. of derivation evaluations)');
19 grid on
20 legend('Euler integrator','Runge Kutta 4 integrator');
21
22 % figure(2)
23 % semilogy(diff_euler, deriv_eval_euler,'b-o', diff_rk4, deriv_eval_rk4,'r-x');
24 % title('Derivative evaluations vs % Difference');
25 % xlabel('% Difference');
26 % ylabel('log(no. of derivation evaluations)');
27 % grid on
28 % legend('Euler integrator','Runge Kutta 4 integrator');
29
30 figure(2)
31 plot(acc_euler, deriv_eval_euler,'b-o', acc_rk4, deriv_eval_rk4,'r-x');
32 title('Derivative evaluations vs % Accuracy');
33 xlabel('% Accuracy');
34 ylabel('log(no. of derivation evaluations)');
35 grid on
36 legend('Euler integrator','Runge Kutta 4 integrator');
```

4.2 Transformation

Following is the script for Transformation from Kepler to Cartesian co-ordinates. Taken from Assignment 1. Previously verified to be accurate.

```
1  %% Transformation algorithm from Kepler to cartesian coordinate frame
2  % Eccentric Anomaly E for elliptical orbit, since e <1
3  err = 1; % initializing difference between previous and current E
4  tol = 1*10^-20; % Tolerance for converge of E
5  E_i0 = M; % initialise E
6  step = 0; % Step number
7  while err > tol
8          step = step +1;
9          E_i1 = E_i0 + ( M - E_i0 + e*sin(E_i0))/( 1 - e*cos(E_i0));
10         err = norm(E_i1 - E_i0);
11         E_i0 = E_i1;
12         E = E_i0; % Update the final value of E
13 end
14 E_deg = rad2deg(E); % Eccentric anomaly in [deg]
15 % Since e<1 theta follows from the kepler relation of an elliptical orbit
16
```

```
n = sqrt(mu/a^3); % Mean motion [rad/s]
theta = 2*atan( sqrt( (1+e)/(1-e) )*tan(E/2)); % True anomaly [rad]
theta_deg = rad2deg(theta); % True anomaly [deg]
r = a*(1-e*cos(E)); % norm of position vector [m]

% Transformation variables :
l1 = cos(Omega)*cos(omega) - sin(Omega)*sin(omega)*cos(i);
l2 = -cos(Omega)*sin(omega) -sin(Omega)*cos(omega)*cos(i);

m1 = sin(Omega)*cos(omega) + cos(Omega)*sin(omega)*cos(i);
m2 = -sin(Omega)*sin(omega) + cos(Omega)*cos(omega)*cos(i);

n1 = sin(omega)*sin(i);
n2 = cos(omega)*sin(i);

% Transformation matrix to convert Kepler position to cartesian position

% [eps,eta] is fixed on Earth, eps is the x axis and pointing to the right
% and eta the y axis point up on the orbital plane(as viewed from the top)
% Similar to Lecture slide 17 of Basic lecture.

eps = r*cos(theta);
eta = r*sin(theta);

T = [ l1 l2; m1 m2; n1 n2];
pos_cartesian = T* [eps; eta];

x = pos_cartesian(1); % x position [m]
y = pos_cartesian(2); % y position [m]
z = pos_cartesian(3); % z position [m]

% Angular momentum H
H = sqrt( mu * a*( 1 - e^2));

% Cartesian velocity parameters in [m/s]
x_dot = (mu/H)*( -l1*sin(theta) + l2*(e + cos(theta))); % x velocity component [m/s]
y_dot = (mu/H)*( -m1*sin(theta) + m2*(e + cos(theta))); % y velocity component [m/s]
z_dot = (mu/H)*( -n1*sin(theta) + n2*(e + cos(theta))); % z velocity component [m/s]
```

Following is the script for Transformation from Cartesian to Kepler elements a and e. Taken from Assignment 1. Previously verified to be accurate.

```
%% Derivation of values of semi-major axis and eccentricity (a,e) to verify the estimations
% Using the script from assignment 1

x = x0(1) ;% [m]
y = x0(2); % [m]
z = x0(3); % [m]

xdot = x0(4); % [m/s]
ydot = x0(5); % [m/s]
zdot = x0(6); % [m/s]

r_vec = [x;y;z]; % position vector [m/s]

V_vec = [xdot; ydot; zdot]; % velocity vector [m/s]

r = norm(r_vec); % position norm [m]
V = norm(V_vec); % velocity norm [m/s]

h = cross(r_vec,V_vec); % angular momentum [m^2/s]
N = cross([0; 0; 1],h); % vector to ascending node [m]

a = 1/ ((2/r) - ( (V^2)/(mu) )); % semi major axis [m]

e_vec = (1/mu)*cross(V_vec,h) - (1/r)*r_vec ; % Eccentricity vector
e = norm(e_vec); % norm of eccentricity
```

4.3 Script INTEG 3

Following is the script to conduct test on the Euler Script:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% AE4878 Mission Geometry and Orbit Design %%%%%%%%%%%%%
%%%%% Assignment 3 - Week 4 - Integrators - Version 1 %%%%%
%%%%% Author Info: Ali Nawaz; Student ID - 4276477 %%%%%%%%
%%%%% Assigned tasks : INTEG-1, INTEG - 3 %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% ////////////// INTEG - 3 //////////////

%% ##### Conversion from Kepler orbit to cartesian components#####
% Script from Assignment 1:

%% Test data
a = 7378.137*10^(3); % Semi major axis [m]
e = 0.1; % Eccentricity
i = 0.0; % inclination angle [rad]
Omega = 0.0; % Right Ascension of Ascending Node [rad]
omega = 0.0;% Argument of pericenter [rad]
M = 0.0; % Mean anomaly [rad]
mu = 3.98600441*10^(14); % [m^3 s^-2] Standard gravitational parameter of Earth from Intro ...
    Lecture ;)

%% Transformation algorithm from Kepler to cartesian coordinate frame
% Eccentric Anomaly E for elliptical orbit, since e <1
err = 1; % initializing difference between previous and current E
tol = 1*10^-20; % Tolerance for converge of E
E_i0 = M; % initialise E
step = 0; % Step number
while err > tol
       step = step +1;
       E_i1 = E_i0 + ( M - E_i0 + e*sin(E_i0))/( 1 - e*cos(E_i0));
       err = norm(E_i1 - E_i0);
       E_i0 = E_i1;
       E = E_i0; % Update the final value of E
end
E_deg = rad2deg(E); % Eccentric anomaly in [deg]
% Since e<1 theta follows from the kepler relation of an elliptical orbit

n = sqrt(mu/a^3); % Mean motion [rad/s]
theta = 2*atan( sqrt( (1+e)/(1-e) )*tan(E/2)); % True anomaly [rad]
theta_deg = rad2deg(theta); % True anomaly [deg]
r = a*(1-e*cos(E)); % norm of position vector [m]

% Transformation variables :
l1 = cos(Omega)*cos(omega) - sin(Omega)*sin(omega)*cos(i);
l2 = -cos(Omega)*sin(omega) -sin(Omega)*cos(omega)*cos(i);

m1 = sin(Omega)*cos(omega) + cos(Omega)*sin(omega)*cos(i);
m2 = -sin(Omega)*sin(omega) + cos(Omega)*cos(omega)*cos(i);

n1 = sin(omega)*sin(i);
n2 = cos(omega)*sin(i);

% Transformation matrix to convert Kepler position to cartesian position

% [eps,eta] is fixed on Earth, eps is the x axis and pointing to the right
% and eta the y axis point up on the orbital plane(as viewed from the top)
% Similar to Lecture slide 17 of Basic lecture.

eps = r*cos(theta);
eta = r*sin(theta);

T = [ l1 l2; m1 m2; n1 n2];
pos_cartesian = T* [eps; eta];

x = pos_cartesian(1); % x position [m]
y = pos_cartesian(2); % y position [m]
z = pos_cartesian(3); % z position [m]

% Angular momentum H
H = sqrt( mu * a*( 1 - e^2));

% Cartesian velocity parameters in [m/s]
x_dot = (mu/H)*( -l1*sin(theta) + l2*(e + cos(theta))); % x velocity component [m/s]
y_dot = (mu/H)*( -m1*sin(theta) + m2*(e + cos(theta))); % y velocity component [m/s]
z_dot = (mu/H)*( -n1*sin(theta) + n2*(e + cos(theta))); % z velocity component [m/s]
```

```
76
77 %% ##### 1.  #####
78 %% Verifying Euler integration code logic with lecture slides
79 x0 = [ x y z x_dot y_dot z_dot]'; % initial state vector
80
81 dt = 10; % time step [s]
82 t0 = 0; % initial time [s]
83 tf = 20; % final time [s]
84
85 res = [x0]; % store results of Euler integration here.
86
87 % Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
88 for t = t0+dt:dt:tf
89     t;
90     r = norm(x0(1:3,1));
91     phi = [ x0(4:6,1) ; (-mu/(r^3)).*x0(1:3,1)]; % incremental function
92     x = x0 + dt.*phi; % Euler integration
93     res = [res, x]; % Storing results row1 : pos, row2: vel, col: iteration number + 1
94     x0 = x; % Updating initial state for next iteration
95 end
96
97 %% Derivation of values of semi-major axis and eccentricity (a,e) to verify the estimations
98 % Using the script from assignment 1
99
100 x = x0(1) ;% [m]
101 y = x0(2); % [m]
102 z = x0(3); % [m]
103
104 xdot = x0(4); % [m/s]
105 ydot = x0(5); % [m/s]
106 zdot = x0(6); % [m/s]
107
108 r_vec = [x;y;z]; % position vector [m/s]
109
110 V_vec = [xdot; ydot; zdot]; % velocity vector [m/s]
111
112 r = norm(r_vec); % position norm [m]
113 V = norm(V_vec); % velocity norm [m/s]
114
115 h = cross(r_vec,V_vec); % angular momentum [m^2/s]
116 N = cross([0; 0; 1],h); % vector to ascending node [m]
117
118 a = 1/ ((2/r) - ( (V^2)/(mu) )); % semi major axis [m]
119
120 e_vec = (1/mu)*cross(V_vec,h) - (1/r)*r_vec ; % Eccentricity vector
121 e = norm(e_vec); % norm of eccentricity
122
123 %% [ VERIFIED, THE ABOVE SCRIPTS WORK PERFECTLY FOR THE VALUES PRESENTED IN THE LECTURE SLIDES]
```

Following is the script used for Cartesian parameter estimation with Euler integrator.

```
1  %% ##### Conversion from Kepler orbit to cartesian components#####
2  % Script from Assignment 1:
3  %% Assignment data
4  height_GEO = 35786*10^(3); % GEO satellite altitude [m] Source: ...
       en.wikipedia.org/wiki/Geostationary_orbit
5  radius_earth = 6378.136*10^(3); % Radius of Earth [m] Source: Intro lecture slide 16
6  a = height_GEO + radius_earth; % Semi major axis [m]
7  e = 0.0; % Eccentricity
8  i = 0.0; % inclination angle [rad]
9  Omega = 0.0; % Right Ascension of Ascending Node [rad]
10 omega = 0.0;% Argument of pericenter [rad]
11 M = 0.0; % Mean anomaly [rad]
12 mu = 3.98600441*10^(14); % [m^3 s^-2] Standard gravitational parameter of Earth from Intro ...
       Lecture ;)
13
14 %% Transformation algorithm from Kepler to cartesian coordinate frame
15 % Eccentric Anomaly E for elliptical orbit, since e <1
16 err = 1; % initializing difference between previous and current E
17 tol = 1*10^-20; % Tolerance for converge of E
18 E_i0 = M; % initialise E
19 step = 0; % Step number
20 while err > tol
21         step = step +1;
22         E_i1 = E_i0 + ( M - E_i0 + e*sin(E_i0))/( 1 - e*cos(E_i0));
```

```
        err = norm(E_i1 - E_i0);
        E_i0 = E_i1;
        E = E_i0; % Update the final value of E
end
E_deg = rad2deg(E); % Eccentric anomaly in [deg]
% Since e<1 theta follows from the kepler relation of an elliptical orbit

n = sqrt(mu/a^3); % Mean motion [rad/s]
theta = 2*atan( sqrt( (1+e)/(1-e) )*tan(E/2)); % True anomaly [rad]
theta_deg = rad2deg(theta); % True anomaly [deg]
r = a*(1-e*cos(E)); % norm of position vector [m]

% Transformation variables :
l1 = cos(Omega)*cos(omega) - sin(Omega)*sin(omega)*cos(i);
l2 = -cos(Omega)*sin(omega) -sin(Omega)*cos(omega)*cos(i);

m1 = sin(Omega)*cos(omega) + cos(Omega)*sin(omega)*cos(i);
m2 = -sin(Omega)*sin(omega) + cos(Omega)*cos(omega)*cos(i);

n1 = sin(omega)*sin(i);
n2 = cos(omega)*sin(i);

% Transformation matrix to convert Kepler position to cartesian position

% [eps,eta] is fixed on Earth, eps is the x axis and pointing to the right
% and eta the y axis point up on the orbital plane(as viewed from the top)
% Similar to Lecture slide 17 of Basic lecture.

eps = r*cos(theta);
eta = r*sin(theta);

T = [ l1 l2; m1 m2; n1 n2];
pos_cartesian = T* [eps; eta];

x = pos_cartesian(1); % x position [m]
y = pos_cartesian(2); % y position [m]
z = pos_cartesian(3); % z position [m]

% Angular momentum H
H = sqrt( mu * a*( 1 - e^2));

% Cartesian velocity parameters in [m/s]
x_dot = (mu/H)*( -l1*sin(theta) + l2*(e + cos(theta))); % x velocity component [m/s]
y_dot = (mu/H)*( -m1*sin(theta) + m2*(e + cos(theta))); % y velocity component [m/s]
z_dot = (mu/H)*( -n1*sin(theta) + n2*(e + cos(theta))); % z velocity component [m/s]
%% ##### 1. EULER INTEGRATION #####

x0 = [ x y z x_dot y_dot z_dot]'; % initial state vector position [x,y,z] [m], velocity ...
    [x_dot, y_dot, z_dot] [m/s]

dt = 50; % time step [s]
t0 = 0; % initial time [s]
sol_day = 24*60*60; % Solar day = 24 hours [s]
tf = sol_day*30; % final time: 30 solar days[s]

res = [x0]; % store results of Euler integration here.

% Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
for t = t0+dt:dt:tf
    t;
    r = norm(x0(1:3,1));
    phi = [ x0(4:6,1) ; (-mu/(r^3)).*x0(1:3,1)]; % incremental function
    x = x0 + dt.*phi; % Euler integration
    res = [res, x]; % Storing results row1 : pos, row2: vel, col: iteration number + 1
    x0 = x; % Updating initial state for next iteration
end

%% Derivation of values of semi-major axis and eccentricity (a,e) to verify the estimations
% Using the script from assignment 1

x = x0(1) ;% [m]
y = x0(2); % [m]
z = x0(3); % [m]

xdot = x0(4); % [m/s]
ydot = x0(5); % [m/s]
```

```
zdot = x0(6); % [m/s]

r_vec = [x;y;z]; % position vector [m/s]

V_vec = [xdot; ydot; zdot]; % velocity vector [m/s]

r = norm(r_vec); % position norm [m]
V = norm(V_vec); % velocity norm [m/s]

h = cross(r_vec,V_vec); % angular momentum [m^2/s]
N = cross([0; 0; 1],h); % vector to ascending node [m]

a = 1/ ((2/r) - ( (V^2)/(mu) )); % semi major axis [m]

e_vec = (1/mu)*cross(V_vec,h) - (1/r)*r_vec ; % Eccentricity vector
e = norm(e_vec); % norm of eccentricity
```

Following is the script used to conduct test on RK4 integrator:

```
%% ##### Conversion from Kepler orbit to cartesian components#####
% Script from Assignment 1:

%% Test data
a = 7378.137*10^(3); % Semi major axis [m]
e = 0.1; % Eccentricity
i = 0.0; % inclination angle [rad]
Omega = 0.0; % Right Ascension of Ascending Node [rad]
omega = 0.0;% Argument of pericenter [rad]
M = 0.0; % Mean anomaly [rad]
mu = 3.98600441*10^(14); % [m^3 s^-2] Standard gravitational parameter of Earth from Intro ...
    Lecture ;)
%% Transformation algorithm from Kepler to cartesian coordinate frame
% Eccentric Anomaly E for elliptical orbit, since e <1
err = 1; % initializing difference between previous and current E
tol = 1*10^-20; % Tolerance for converge of E
E_i0 = M; % initialise E
step = 0; % Step number
while err > tol
       step = step +1;
       E_i1 = E_i0 + ( M - E_i0 + e*sin(E_i0))/( 1 - e*cos(E_i0));
       err = norm(E_i1 - E_i0);
       E_i0 = E_i1;
       E = E_i0; % Update the final value of E
end
E_deg = rad2deg(E); % Eccentric anomaly in [deg]
% Since e<1 theta follows from the kepler relation of an elliptical orbit

n = sqrt(mu/a^3); % Mean motion [rad/s]
theta = 2*atan( sqrt( (1+e)/(1-e) )*tan(E/2)); % True anomaly [rad]
theta_deg = rad2deg(theta); % True anomaly [deg]
r = a*(1-e*cos(E)); % norm of position vector [m]

% Transformation variables :
l1 = cos(Omega)*cos(omega) - sin(Omega)*sin(omega)*cos(i);
l2 = -cos(Omega)*sin(omega) -sin(Omega)*cos(omega)*cos(i);

m1 = sin(Omega)*cos(omega) + cos(Omega)*sin(omega)*cos(i);
m2 = -sin(Omega)*sin(omega) + cos(Omega)*cos(omega)*cos(i);

n1 = sin(omega)*sin(i);
n2 = cos(omega)*sin(i);

% Transformation matrix to convert Kepler position to cartesian position

% [eps,eta] is fixed on Earth, eps is the x axis and pointing to the right
% and eta the y axis point up on the orbital plane(as viewed from the top)
% Similar to Lecture slide 17 of Basic lecture.

eps = r*cos(theta);
eta = r*sin(theta);

T = [ l1 l2; m1 m2; n1 n2];
pos_cartesian = T* [eps; eta];

x = pos_cartesian(1); % x position [m]
```

```
y = pos_cartesian(2); % y position [m]
z = pos_cartesian(3); % z position [m]

% Angular momentum H
H = sqrt( mu * a*( 1 - e^2));

% Cartesian velocity parameters in [m/s]
x_dot = (mu/H)*( -l1*sin(theta) + l2*(e + cos(theta))); % x velocity component [m/s]
y_dot = (mu/H)*( -m1*sin(theta) + m2*(e + cos(theta))); % y velocity component [m/s]
z_dot = (mu/H)*( -n1*sin(theta) + n2*(e + cos(theta))); % z velocity component [m/s]
%% Verifying RK4 integration code logic with lecture slides
% initial state vector
x0 = [ x y z x_dot y_dot z_dot]'; % initial position

dt = 10; % time step [s]
t0 = 0; % initial time [s]
tf = 20; % final time [s]

res = [x0]; % store results of Euler integration here.

% Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
for t = t0+dt:dt:tf

    k1 = [ x0(4:6,1) ; (-mu/(r^3)).*x0(1:3,1)];

    x0_k2 = x0 + (dt/2).*k1;
    k2 = [ x0_k2(4:6,1) ; (-mu/(r^3)).*x0_k2(1:3,1)];

    x0_k3 = x0 + (dt/2).*k2;
    k3 = [ x0_k3(4:6,1) ; (-mu/(r^3)).*x0_k3(1:3,1)];

    x0_k4 = x0 + dt.*k3;
    k4 = [ x0_k4(4:6,1) ; (-mu/(r^3)).*x0_k4(1:3,1)];

    phi = (1/6).* (k1 + 2.*k2 + 2.*k3 + k4); % incremental function

    x = x0 + dt.*phi; % RK4 integration
    res = [res, x]; % Storing results row 1 : pos, row2: vel, col: iteration number + 1
    x0 = x; % Updating initial state for next iteration
end

%% Derivation of values of semi-major axis and eccentricity (a,e) to verify the estimations
% Using the script from assignment 1

x = x0(1) ;% [m]
y = x0(2); % [m]
z = x0(3); % [m]

xdot = x0(4); % [m/s]
ydot = x0(5); % [m/s]
zdot = x0(6); % [m/s]

r_vec = [x;y;z]; % position vector [m/s]

V_vec = [xdot; ydot; zdot]; % velocity vector [m/s]

r = norm(r_vec); % position norm [m]
V = norm(V_vec); % velocity norm [m/s]

h = cross(r_vec,V_vec); % angular momentum [m^2/s]
N = cross([0; 0; 1],h); % vector to ascending node [m]

a = 1/ ((2/r) - ( (V^2)/(mu) )); % semi major axis [m]

e_vec = (1/mu)*cross(V_vec,h) - (1/r)*r_vec ; % Eccentricity vector
e = norm(e_vec); % norm of eccentricity

% [ VERIFIED, THE ABOVE SCRIPTS WORK PERFECTLY FOR THE VALUES PRESENTED IN
% THE LECTURE SLIDES]
```

Following is the script used for Cartesian parameter estimation with RK4 integrator.

```
%% ##### Conversion from Kepler orbit to cartesian components#####
% Script from Assignment 1:
%% Assignment data
```

```
height_GEO = 35786*10^(3); % GEO satellite altitude [m] Source: ...
    en.wikipedia.org/wiki/Geostationary_orbit
radius_earth = 6378.136*10^(3); % Radius of Earth [m] Source: Intro lecture slide 16
a = height_GEO + radius_earth; % Semi major axis [m]
e = 0.0; % Eccentricity
i = 0.0; % inclination angle [rad]
Omega = 0.0; % Right Ascension of Ascending Node [rad]
omega = 0.0;% Argument of pericenter [rad]
M = 0.0; % Mean anomaly [rad]
mu = 3.98600441*10^(14); % [m^3 s^-2] Standard gravitational parameter of Earth from Intro ...
    Lecture ;)

%% Transformation algorithm from Kepler to cartesian coordinate frame
% Eccentric Anomaly E for elliptical orbit, since e <1
err = 1; % initializing difference between previous and current E
tol = 1*10^-20; % Tolerance for converge of E
E_i0 = M; % initialise E
step = 0; % Step number
while err > tol
       step = step +1;
       E_i1 = E_i0 + ( M - E_i0 + e*sin(E_i0))/( 1 - e*cos(E_i0));
       err = norm(E_i1 - E_i0);
       E_i0 = E_i1;
       E = E_i0; % Update the final value of E
end
E_deg = rad2deg(E); % Eccentric anomaly in [deg]
% Since e<1 theta follows from the kepler relation of an elliptical orbit

n = sqrt(mu/a^3); % Mean motion [rad/s]
theta = 2*atan( sqrt( (1+e)/(1-e) )*tan(E/2)); % True anomaly [rad]
theta_deg = rad2deg(theta); % True anomaly [deg]
r = a*(1-e*cos(E)); % norm of position vector [m]

% Transformation variables :
l1 = cos(Omega)*cos(omega) - sin(Omega)*sin(omega)*cos(i);
l2 = -cos(Omega)*sin(omega) -sin(Omega)*cos(omega)*cos(i);

m1 = sin(Omega)*cos(omega) + cos(Omega)*sin(omega)*cos(i);
m2 = -sin(Omega)*sin(omega) + cos(Omega)*cos(omega)*cos(i);

n1 = sin(omega)*sin(i);
n2 = cos(omega)*sin(i);

% Transformation matrix to convert Kepler position to cartesian position

% [eps,eta] is fixed on Earth, eps is the x axis and pointing to the right
% and eta the y axis point up on the orbital plane(as viewed from the top)
% Similar to Lecture slide 17 of Basic lecture.

eps = r*cos(theta);
eta = r*sin(theta);

T = [ l1 l2; m1 m2; n1 n2];
pos_cartesian = T* [eps; eta];

x = pos_cartesian(1); % x position [m]
y = pos_cartesian(2); % y position [m]
z = pos_cartesian(3); % z position [m]

% Angular momentum H
H = sqrt( mu * a*( 1 - e^2));

% Cartesian velocity parameters in [m/s]
x_dot = (mu/H)*( -l1*sin(theta) + l2*(e + cos(theta))); % x velocity component [m/s]
y_dot = (mu/H)*( -m1*sin(theta) + m2*(e + cos(theta))); % y velocity component [m/s]
z_dot = (mu/H)*( -n1*sin(theta) + n2*(e + cos(theta))); % z velocity component [m/s]
%% ##### 2. APPLICATION OF RK4 #####
% initial state vector
x0 = [ x y z x_dot y_dot z_dot]'; % initial position

dt = 50; % time step [s]
t0 = 0; % initial time [s]
sol_day = 24*60*60; % Solar day = 24 hours [s]
tf = sol_day*30; % final time: 30 solar days[s]

res = [x0]; % store results of Euler integration here.
```

```
78
79  % Initial step at t = 0 is predefined, iteration runs from t0+dt to tf
80  for t = t0+dt:dt:tf
81
82      k1 = [ x0(4:6,1) ; (-mu/(r^3)).*x0(1:3,1)];
83
84      x0_k2 = x0 + (dt/2).*k1;
85      k2 = [ x0_k2(4:6,1) ; (-mu/(r^3)).*x0_k2(1:3,1)];
86
87      x0_k3 = x0 + (dt/2).*k2;
88      k3 = [ x0_k3(4:6,1) ; (-mu/(r^3)).*x0_k3(1:3,1)];
89
90      x0_k4 = x0 + dt.*k3;
91      k4 = [ x0_k4(4:6,1) ; (-mu/(r^3)).*x0_k4(1:3,1)];
92
93      phi = (1/6).* (k1 + 2.*k2 + 2.*k3 + k4); % incremental function
94
95      x = x0 + dt.*phi; % RK4 integration
96      res = [res, x]; % Storing results row 1 : pos, row2: vel, col: iteration number + 1
97      x0 = x; % Updating initial state for next iteration
98  end
99
100 %% Derivation of values of semi-major axis and eccentricity (a,e) to verify the estimations
101 % Using the script from assignment 1
102
103 x = x0(1) ;% [m]
104 y = x0(2); % [m]
105 z = x0(3); % [m]
106
107 xdot = x0(4); % [m/s]
108 ydot = x0(5); % [m/s]
109 zdot = x0(6); % [m/s]
110
111 r_vec = [x;y;z]; % position vector [m/s]
112
113 V_vec = [xdot; ydot; zdot]; % velocity vector [m/s]
114
115 r = norm(r_vec); % position norm [m]
116 V = norm(V_vec); % velocity norm [m/s]
117
118 h = cross(r_vec,V_vec); % angular momentum [m^2/s]
119 N = cross([0; 0; 1],h); % vector to ascending node [m]
120
121 a = 1/ ((2/r) - ( (V^2)/(mu) )); % semi major axis [m]
122
123 e_vec = (1/mu)*cross(V_vec,h) - (1/r)*r_vec ; % Eccentricity vector
124 e = norm(e_vec); % norm of eccentricity
```

Following script is used to provide insight into the accuracies of Euler and Runge Kutta 4 integrator to facilitate trade off.

```
1  %% Exercise 4
2
3  % Accumulated data after multiple runs
4
5  step_sizes = [ 5000 500 50 5 2.5];
6  deriv_eval_euler = [518 5184 51840 518400 1036800]; % No. of derivative evaluations for Euler
7  deriv_eval_rk4 = 4.*deriv_eval_euler; % No. of derivative evaluations for RK4
8
9  acc_euler = [ -3248.5272 -150.9511 27.4489 87.7726  93.5354]; % Percentage accuracy Euler
10 acc_rk4  = [100 100 100 100  100]; % Percentage accuracy Runge Kutta 4
11
12 diff_euler = 100.-acc_euler;
13 diff_rk4 = 100.- acc_rk4;
14 figure(1)
15 semilogy(acc_euler, deriv_eval_euler,'b-o', acc_rk4, deriv_eval_rk4,'r-x');
16 title('Derivative evaluations vs % Accuracy');
17 xlabel('% Accuracy');
18 ylabel('log(no. of derivation evaluations)');
19 grid on
20 legend('Euler integrator','Runge Kutta 4 integrator');
21
22 % figure(2)
23 % semilogy(diff_euler, deriv_eval_euler,'b-o', diff_rk4, deriv_eval_rk4,'r-x');
```

```
% title('Derivative evaluations vs % Difference');
% xlabel('% Difference');
% ylabel('log(no. of derivation evaluations)');
% grid on
% legend('Euler integrator','Runge Kutta 4 integrator');

figure(2)
plot(acc_euler, deriv_eval_euler,'b-o', acc_rk4, deriv_eval_rk4,'r-x');
title('Derivative evaluations vs % Accuracy');
xlabel('% Accuracy');
ylabel('log(no. of derivation evaluations)');
grid on
legend('Euler integrator','Runge Kutta 4 integrator');
```

References

[1] Fundamentals of Astrodynamics - K. Wakker. https://www.researchgate.net/publication/272507882_Fundamentals_of_Astrodynamics.

[2] R. Noomen. *AE4878 Mission Geometry and Orbit Design*. TU Delft, 2017.