AE4879 Mission Geometry and Orbit Design

# Assignment 3: Integrators
## INTEG-1 and INTEG-3

Simon Billemont

September 24, 2010

# Assignment 3: Integrators

This assignment deals with different integrators (Euler and RK4) for which two cases are discussed. The first case is a simple linear motion with constant acceleration. In the second case, the different integrators are applied to specific a orbit propagation problem (GEO orbit).

# 1. Integrators

Integrators are a piece of software to approximate the initial value problems for a set of given ordinary differential equations. This is done sequentially by computing a new value from the previous values and a function to describe the changes currently happening (slope, derivative). This is given mathematical sense in eq 1 (From [1] slide 12).

$$\mathbf{x}(t_0 + h) \approx \mathbf{x}(t_0) + h\,\mathbf{\Phi} \tag{1}$$

Where x is a function value at a given time, $t_0$ the time of the last function value, h the time step and $\Phi$ is the increment function. The increment function differs between integration schemes. In this document two different integrators are discussed namely the Euler and Runge - Kutta ($4^{th}$ order) integrators.

## 1.1. Euler integrator

The Euler integrator is a very simple integrator where the increment function ($\Phi$) is basically the derivative of the function that is to be approximated (see eq 2, from [1] slide 12).

$$\mathbf{\Phi}_{Euler} = \dot{\mathbf{x}}_0 = \mathbf{f}(t_0, \mathbf{x}_0) \tag{2}$$

What this means is that any change in the function after the last point is neglected. The result of this is that variable functions (and most functions are) cannot be followed correctly. For the source code, see appendix A and for a test case see appendix A.

## 1.2. Runge - Kutta integrator ($4^{th}$ order)

The Runge - Kutta integrator (of the $4^{th}$ order, RK4) has a more complex $\Phi$ function, but is also computationally heavier. The increment function is defined is the weighted average of the derivative at 4 different positions (see eq 3, from [1] slide 17)

$$\mathbf{\Phi}_{RK4} = \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{3}$$

With $k_1$ to $k_4$ the derivatives of the function of interest. They can be found using eq 4-7 (From [1] slide 17).

$$\mathbf{k}_1 = \dot{f}(t_0,\, \mathbf{x}_0) \tag{4}$$

$$\mathbf{k}_2 = \dot{f}(t_0 + h/2,\, \mathbf{x}_0 + h\mathbf{k}_1/2) \tag{5}$$

$$\mathbf{k}_3 = \dot{f}(t_0 + h/2,\, \mathbf{x}_0 + h\mathbf{k}_2/2) \tag{6}$$

$$\mathbf{k}_4 = \dot{f}(t_0 + h,\, \mathbf{x}_0 + h\mathbf{k}_3) \tag{7}$$

This means that there are 4 times the derivative calculation with respect to the Euler integrator. However RK4 can follow functions a lot better resulting in good approximations with step sizes a lot higher than when using Euler. For code specific comments, see appendix A and for a test case see appendix A.

# 2. Linear motion (INTEG-1)

In this section an object with a linear motion and constant acceleration is considered. The problem is the to find the position and velocity of the object after a given time. This is done both analytically and numerically. This allows for the comparison of the two different integrators and find there respective errors. The initial conditions are given below.

$$\begin{aligned} a_0 &= 2\,[m/s^2] \\ v_0 &= 0\,[m/s] \\ r_0 &= 0\,[m] \end{aligned}$$

To solve the problem numerically, one can simply integrate the acceleration twice and get both the position and velocity expressions (see eq 8-10).

$$a = a_0 \tag{8}$$

$$v = \int a = a_0 \cdot t + c_1 \tag{9}$$

$$r = \int v = a_0 \cdot \frac{t^2}{2} + c_1 \cdot t + c_2 \tag{10}$$

Using the initial conditions one finds that the terms containing $c_1$ and $c_2$ vanish. For the integrators, a set of ordinary differential equations (ODE) are required. If equation 10 is inverted, one second order differential equation is formed. Therefore the found equation of motion has to split up into two ODEs (eq 11). These two equations then form the derivative function used in $\Phi$ (see eq 12)

$$\frac{\mathrm{dv}}{\mathrm{dt}} = a \text{ and } \frac{\mathrm{dr}}{\mathrm{dt}} = v \tag{11}$$

$$\dot{f}(t, r, v) = \begin{bmatrix} a_0 \\ v \end{bmatrix} \tag{12}$$

When the calculations are performed one finds results as in table 1. It contains the end values for the analytical expressions. Furthermore it holds the end values for the integrators and the error with respect to the analytical outcome.

| | $v(t=60)[m/s]$ | error in $v[m/s]$ | $r(t=60)[m]$ | error in $r[m]$ |
|---|---|---|---|---|
| Analytical | 120 | – | 3600 | – |
| Euler($h=5s$) | 120 | 0 | 3540 | $-300$ |
| Euler($h=1s$) | 120 | 0 | 3540 | $-60$ |
| RK4($h=5$) | 120 | 0 | 3600 | 0 |
| RK4($h=1$) | 120 | 0 | 3600 | 0 |

**Table 1:** Integration results for the linear motion case

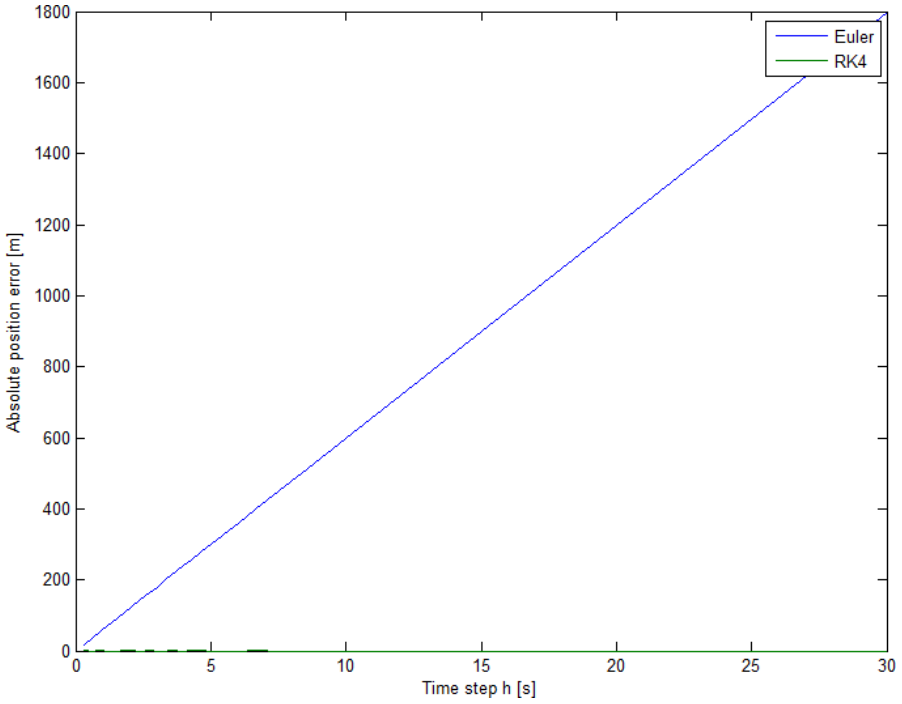


**Figure 1:** Absolute error [m] vs step size h [s] for the linear case

It is clear that velocity is correctly estimated using both integrators. This is due to the fact that the acceleration is constant. Both methods estimate the average slope of the function, and since this is constant, no error is made. Furthermore can one see that the RK4 integrator also doesn't make any error in the position estimate. In order to gain insight in the evolution of the accuracy of both methods, a plot was made of the error vs step size (see fig 1).

From this figure one can see that the Euler integrator has hard time to get the correct answer. The step size I felt comfortable with for the Euler integrator was about 1s. At that step size, the error with respect to the true value is about 1.5% of the total distance traveled. For the RK4 intergrator, there seems to be no effect of step size, as the result is always dead on. However this is most likely a special case. The best integrator for this type of motion is clearly the RK4 integrator.

# 3. Geostationary orbit (INTEG-3)

In this section the motion of a geostationary satellite is considered. The problem is the to find the propagation of the satellite after 1 month (30 days). This is done by computing its Cartesian state vector and propagating that trough time. Then the orbit semi-major axis and eccentricity are recomputed and compared to the start values. The initial conditions are given below.

$$\begin{gathered} a_0 = 42164000\,[m] \\ e_0 = 0\,[-] \\ i_0 = 0\,[deg] \\ \omega_0 = 0\,[deg] \\ \Omega_0 = 0\,[deg] \\ M_0 = 0\,[deg] \end{gathered}$$

The first step is to find the equations of motion, and formulate those in a set of ODE's. It can be assumed that there is only one force acting on the satellite, namely the gravitational force. All other forces are thus neglected. It is general knowledge that the acceleration due to gravity can be formulated as eq 13(From eg [2]).

$$\frac{d^2r}{dt} = a = -r\frac{\mu}{|r|^3} \tag{13}$$

Note: the relations are all vector relations, and thus are a condensed form of three equations (one for each component). The gravitational parameter used was $\mu = 3.98600441 \cdot 10^{14}\ m^2/s^3$ (Earth, see[2]). The equation of motion can then be decomposed in two ODE's. This results in the same expression as before (eq 11). The only difference is that the acceleration is not a constant but eq 13. The two equations can then be combined to form the derivative function (see eq 14)

$$\dot{f}(t, r, v) = \begin{bmatrix} v \\ -r \cdot \frac{\mu}{|r|^3} \end{bmatrix} \tag{14}$$

When the integration is done with both the Euler and RK4 integrators, a large difference in accuracies are observed. A visualization of the orbits is given in fig 2

The integration for an accuracy of 3m/month (what I think to be be a good step size for the integrators, see last paragraph below for explanation) is summarized in table 2. At first glance one can see that the Euler integrator converges so slow, that it is not usable for this simulation. The RK4 integrator however does seem to do a good job, even at large step sizes. This trend is confirmed by the values for the eccentricity.

When the amount of derivatives computed is plotted vs the difference in semi major axis (GEO, so should remain constant), figure 3 is created. Once again, it can be confirmed that the Euler integrator isn't a good choice as integrator. Even at very small step sizes (h=10s or $25 \cdot 10^4$ derivatives) the deviation is 9381km. For the RK4 integrator, a zoomin has been made (see fig 4). One can see that the RK4 integrator is (even at larger step size), considerably better. For largest step size (h=4000s, 2596 derivatives) an error of 500km was computed. This is already better then the best Euler calculations, while being a lot faster to compute 34.09s (Euler) vs 0.25s(RK4).

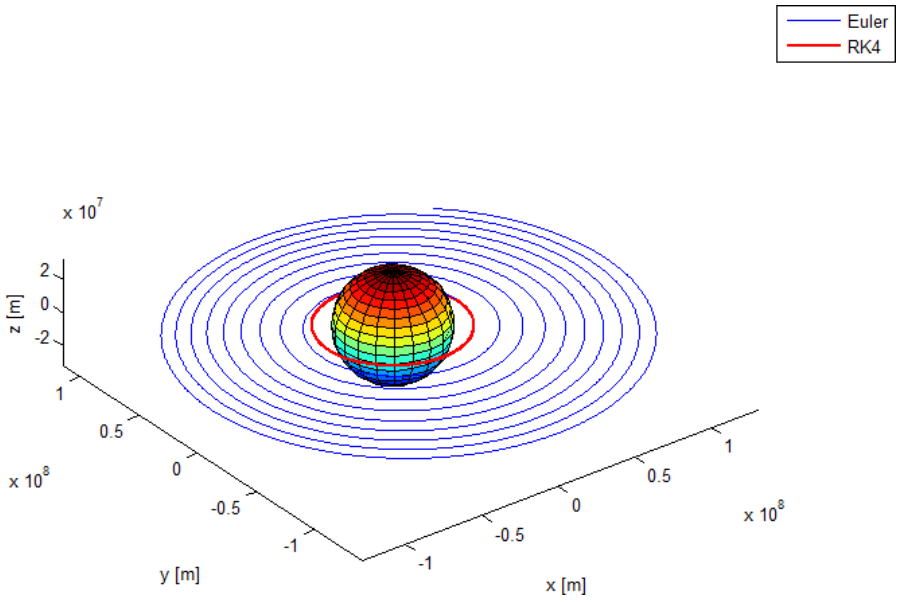


**Figure 2:** Visualization of the GEO orbit propagation (ECI, 30 days), earth is scaled up 5x

| | $a[km]$ | error in $a[km]$ | $e[-]$ | error in $e[0]$ |
|---|---|---|---|---|
| Start value | 42164 | – | 0 | – |
| Euler$(h = 5s)$ | 47319 | 5155 | 0.0011 | 0.0011 |
| Euler$(h = 360s)$ | 132522 | 90358 | 0.0863 | 0.0863 |
| RK4$(h = 5s)$ | 42164 | $3 \cdot 10^{-9}$ | 0 | 0 |
| RK4$(h = 360s)$ | 42164 | $-2.76 \cdot 10^{-3}$ | 0 | 0 |

**Table 2:** Integration results for the linear motion case

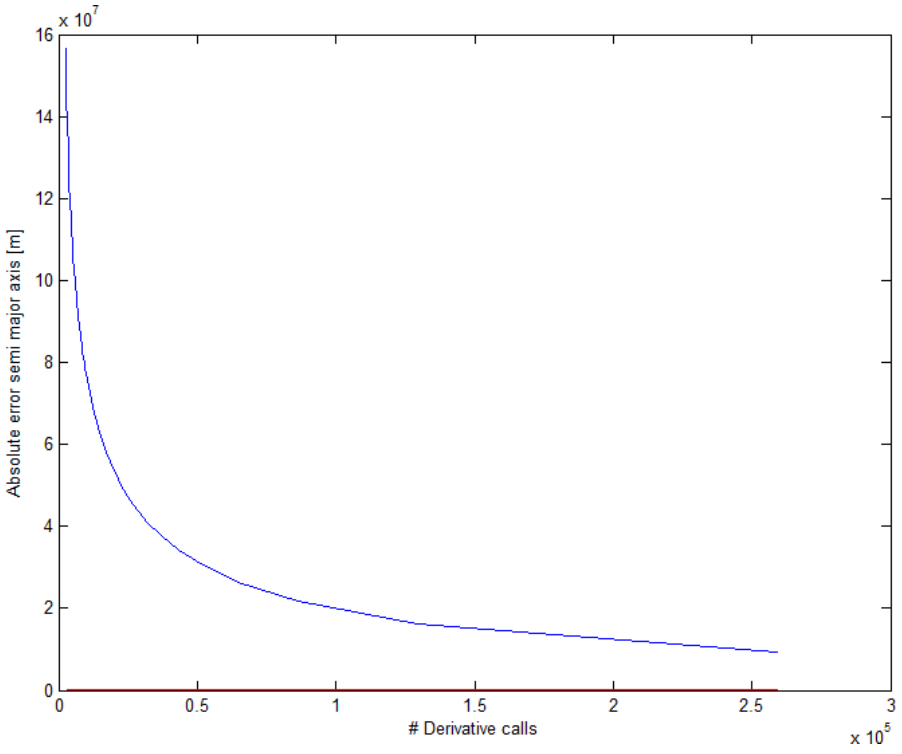


**Figure 3:** Euler (blue) and RK4 (red) derivative calls vs semi-major axis difference [m] for a GEO orbit of 30 days

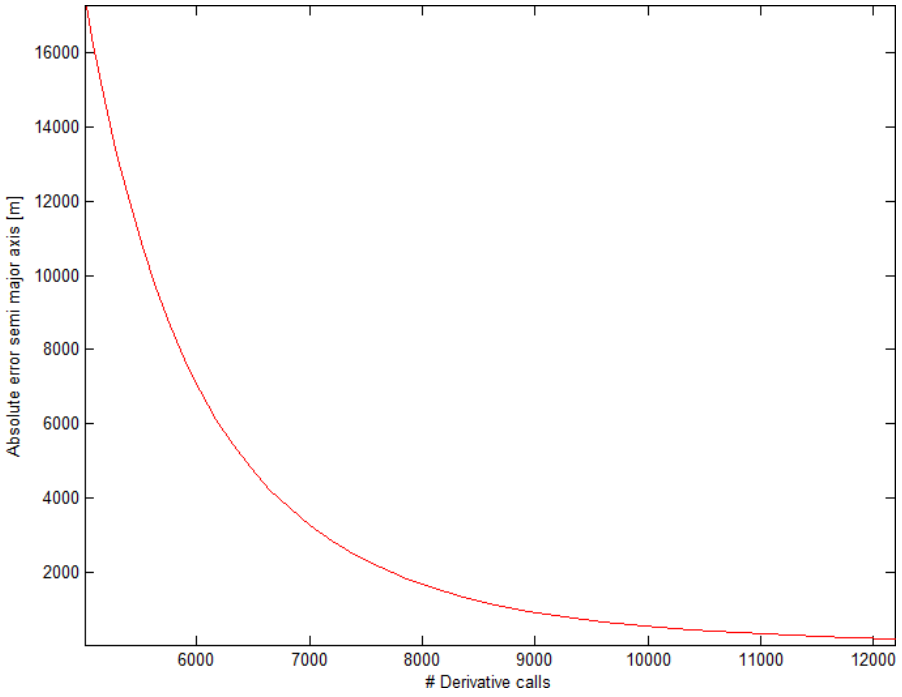


**Figure 4:** RK4 Derivative calls vs semi-major axis difference [m] for a GEO orbit of 30 days

The accuracy that I think would be acceptable over a month time would be about 3m. That amounts to about 36m per year. These deviations are negligible with respect to other perturbation forces acting on satellites in a GEO orbit. This can be achieved with the RK4 integrator with a time step of 6 minutes (28804 derivative calls). An accuracy of 2.75m is then achieved. This is computed for an entire month in 1.98s. However if you cannot verify your results as with a pure Kepler orbit, a smaller step size would be advisable (in the order of about 4 minutes). Important to note is that these accuracies are important when integration times are increased to years and even decades. If only one month is considered, the accuracies can be loosened even further. Think in the range of 15 minutes for RK4 (that amounts to about 30m).

# References

[1] R. Noomen, *AE4-879 Integrators*, TUDelft Lecture Slides, 2010.

[2] J. R. Wertz, *Orbit & Constellation Design & Management*, second printing ed. El Segundo, California: Microcosm Press, 2009.

[3] MathWorks. (2010a) Matlab 7.11. Natick, MA.

[4] R. Rieber. (2006, December) Orbital mechanics library. MATLAB Central File Exchange. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/13439-orbital-mechanics-library

# Additional information

Estimated work time:

$\sim$ 3h Studying theory + $\sim$ 4h making assignment + $\sim$ 5h writing report = $\sim$ 12h

### Made by

Simon Billemont

Stud Nr: 1387855

s.billemont@student.tudelft.nl

### Version history

Version 1: Initial document

# A. Matlab source code

The code to compute the above computations was written in MATLAB 7.11 (2010b)[3]. A structured overview of the dependencies is given below:

- INTEG1.m
  - INTEG1_f.m
  - MyIntegrator.m
    - EulerIntegrator.m
    - Rk4Integrator.m
- INTEG3.m
  - INTEG3_f.m
  - MyIntegrator.m
    - EulerIntegrator.m
    - Rk4Integrator.m
  - Orbital Mechanics Library[4]

Each assignment has a separate script (INTEG.m and INTEG3.m). They reference there derivative function (INTEG1_f.m and INTEG3_f.m) which is passed to the integrator (MyIntegrator.m). MyIntegrator then uses a specific integrator (EulerIntegrator.m and Rk4Integrator.m) to integrate the given derivative function.

Note for INTEG-3 a MATLAB Central File Exchange library was used to convert Keplerian state vector into the Cartesian state vector and visa versa. This was done because the first assignment was made in Mathematica, and the goal of this assignment was integrators, and not state vector conversion.

**Listing 1: INTEG1.m**: Script to compute all the questions of INTEG-1

```
%% Setup environment
clc
clearvars
close all

% Make a new utility for saving pictures
saver = ImSav();
saver.c_plotsDir = '../images/matlab/';

%% Setup constants
t0 = 0;                                 %s
te = 60;                                %s
dt = 5;                                 %s

x0 = 0;                                 %m
v0 = 0;                                 %m/s
a0 = 2;                                 %m/sš

t = t0:dt:te;                           %s

%% Part 1
a_true = a0;                            %m/sš
v_true = vectorize(inline('2*t','t'));  %m/s
r_true = vectorize(inline('2*(t^2/2)','t'));%m

fprintf('End velocity: %f\n', v_true(te));
fprintf('End position: %f\n', r_true(te));

%% Part 2/3
global activeIntegrator integratorCounts;
integratorCounts = [0,0];
```

```
results = struct();
results.true  = [v_true(t); r_true(t)]';
results.euler = zeros(length(t),2);
results.rk4   = zeros(length(t),2);

euler = MyIntegrator(MyIntegrator.EULER);
rk4   = MyIntegrator(MyIntegrator.RK4  );

activeIntegrator = 1;  %#ok<NASGU> ITS GLOBAL
results.euler = euler.Integrate(t, [v0 x0], @INTEG1_f);

activeIntegrator = 2;
results.rk4 = rk4.Integrate(t, [v0 x0], @INTEG1_f);

fprintf('Euler errors: {v:%fm/s\tr:%fm}\n', results.euler(end,:)- results.true(end,:));
fprintf('RK4   errors: {v:%fm/s\tr:%fm}\n', results.rk4(end,:)  - results.true(end,:));
fprintf('Derivative calls: {euler:%d\trk4:%d}\n', integratorCounts);

%% Part 4

% results of the accuractie test [dt, ErrorEuler, ErrorRK4]
rTrue = r_true(te);
accuracies = zeros(20, 3);
for i=1:100
    % Map the index to the time step
    % Makes a timestep that is an integer fraction of te
    % It varies from 0.15s to 15s
    dt = te/round(te/(0.3 * i));
    t  = t0:dt:te;
    results.euler = euler.Integrate(t, [v0 x0], @INTEG1_f);
    results.rk4   = rk4.  Integrate(t, [v0 x0], @INTEG1_f);
    accuracies(i,:) = [dt, results.euler(end,2)-rTrue, results.rk4(end,2)-rTrue];
end

plot(accuracies(:,1),abs(accuracies(:,2:3)))
legend('Euler', 'RK4');
xlabel('Time step h [s]');
ylabel('Absolute position error [m]');
saver.saveImage(gcf,'linearMotion-Error');
```

**Listing 2: INTEG1.m**: Script to compute all the questions of INTEG-3

```
%% Setup environment
clc
clearvars
close all

% Make a new utility for saving pictures
saver = ImSav();
saver.c_plotsDir = '../images/matlab/';

%% Setup constants

mu = 3.98600441E14;                     % mş/sš
Rearth = 6378136;                       % m

t0 = 0;                                 % s
te = 30*24*3600;                        % s
dt = 360;                                 % s

a0     = 42164000;                      % m
e0     = 0;                             % -
i0     = 0;                             % rad
omega0 = 0;                             % rad
raan0  = 0;                             % rad
M0     = 0;                             % rad

% For testing with the values from the sheets
% a0     = 7378137;                         % m
% e0     = 0.1;                             % -
% i0     = 0;                               % rad
% omega0 = 0;                               % rad
% raan0  = 0;                               % rad
% M0     = 0;                               % rad
% t0 = 0;                                   % s
% te = 4;                                   % s
% dt = 2;                                   % s

% I made the first two assignments in Mathematica, and to prevent duplicate
% work, i used a ready and tested MATLBAL orbital library
% see http://www.mathworks.com/matlabcentral/fileexchange/13439-orbital-mechanics-library
% Orbital Mechanics Library by Richard Rieber
[R0 V0] = randv(a0, e0, i0, raan0, omega0, nuFromM(M0, e0), mu);

x0 = [V0; R0]';                         % m/s and m

t = t0:dt:te;                           %s

%% Part 1
results = struct();
results.euler = zeros(length(t),6);
results.rk4   = zeros(length(t),6);
```

```
euler = MyIntegrator(MyIntegrator.EULER);
rk4   = MyIntegrator(MyIntegrator.RK4  );

results.euler = euler.Integrate(t, x0, @INTEG3_f)
results.rk4   = rk4  .Integrate(t, x0, @INTEG3_f)

[a, e] = elorb(results.euler(end,4:6),results.euler(end,1:3), mu);
fprintf('Euler errors: {a:%fm\te:%f}\n', a - a0, e - e0);
[a, e] = elorb(results.rk4(end,4:6),results.rk4(end,1:3), mu);
fprintf('RK4   errors: {a:%fm\te:%f}\n', a - a0, e - e0);
fprintf('Derivative calls: {euler:%d\trk4:%d}\n', length(t),length(t)*4);



plot3(results.euler(:,4),results.euler(:,5),results.euler(:,6), 'b')
hold on
    plot3(results.rk4(:,4),results.rk4(:,5),results.rk4(:,6), 'r', 'linewidth', 2)
    [x,y,z] = sphere;
    scale = Rearth.*5;
    surf(x*scale,y*scale,z*scale);
hold off
axis equal
legend('Euler', 'RK4');
xlabel('x [m]');
ylabel('y [m]');
zlabel('z [m]');
saver.saveImage(gcf,'GEO-visual');

%% Part 4

% results of the accuractie test [dt, ErrorEuler, ErrorRK4]
count = 100;
accuracies = zeros(count, 7);
for i=1:count
    % Map the index to the time step
    % Makes a timestep that is an integer fraction of te
    % It varies from 0.15s to 15s
    dt = te/round(te/(1000*(i/count)));
    t  = t0:dt:te;
    fprintf('Finding accuracy of dt: %f (%d)\n', dt, length(t));
    results.euler = euler.Integrate(t, x0, @INTEG3_f);
    results.rk4   = rk4.  Integrate(t0:dt*4:te, x0, @INTEG3_f);

    [a_euler, e_euler] = elorb(results.euler(end,4:6),results.euler(end,1:3), mu);
    [a_rk4, e_rk4]     = elorb(results.rk4(end,4:6),  results.rk4(end,1:3),   mu);
    accuracies(i,:) = [dt, length(results.euler), a_euler-a0, e_euler-e0,length(results.rk4)*4, a_rk4-a0, e_rk4-e0];
end

plot(accuracies(:,2),abs(accuracies(:,3)))
hold on
    plot(accuracies(:,5),abs(accuracies(:,6)), 'r')
hold off
xlabel('# Derivative calls')
ylabel('Absolute error semi major axis [m]')
```

**Listing 3: INTEG1f.m**: Derivative function used by the integrators for INTEG-1

```
function [ x ] = INTEG1_f( ~, x0 )
%INTEG1_F_V deriviative of the position function (a = 2)
    global activeIntegrator integratorCounts;
    integratorCounts(activeIntegrator) = integratorCounts(activeIntegrator)+1;

    x = [2, ...
         x0(1)];

end
```

**Listing 4: INTEG1f.m**: Derivative function used by the integrators for INTEG-3

```
function [ x ] = INTEG3_f( ~, x0 )
%INTEG1_F deriviative of the catresian state vector
    mu = 3.98600441E14;                    % mş/sš

    % x0 = [Vx Vy Vz Rx Ry Rz]
    V0   = x0(1:3);
    R0   = x0(4:6);

    % a = r * mu/|r|^3
    a    = -R0 * (mu/(norm(R0)^3));

    % x = [dV/dt dR/dt]
    x    = [a     V0];
end
```

**Listing 5: MyIntegrator.m**: links the integration method to the derivative function and performs all the iteration steps

```
classdef MyIntegrator < handle
    properties (Constant)
        EULER = @EulerIntegrator;
        RK4   = @Rk4Integrator;
    end
    properties
        func
        type
    end
    methods
        function [obj] = MyIntegrator(type)
            obj.type = type;
        end
        function [ x ] = Integrate(obj, t, x0, func)
            obj.func = func;

            % debug/progress output interval
            if (length(t) < 1E4)
                infoI = NaN;
            else if (length(t) < 1E7)
                    infoI = round(length(t)/10);
                else
                    infoI = 1E6;
                end
            end

            x = zeros(length(t),size(x0,2));
            x(1,:) = x0;
            % loop over all t and integrate one step
            for i=2:length(t)
                if (mod(i, infoI) == 0)
                    fprintf('Iteration %d (%f%%), t=%f, x:\n', i, i/length(t)*100, t(i-1))
                    fprintf('%f\n',x(i-1, :));
                end
                x(i,:) = obj.type(obj, t, i, x(i-1,:))';
            end
        end

        %compute one integration step
        function [ x ] = Step(obj, t, i, x0, phi)
            dt = t(i) - t(i-1);
            x  = x0(1,:) + dt .* phi(t(i-1), dt, x0);
        end
    end
end
```

**Listing 6: EulerIntegrator.m**: Computes the $\Phi$ function for the Euler integrator

```
function [ x ] = EulerIntegrator(obj, t, idx, x0 )
    function phi = phi(t, dt, x)
        phi = obj.func(t,x);
    end

    x = obj.Step(t, idx, x0, @phi);
end
```

**Listing 7: Rk4Integrator.m**: Computes the $\Phi$ function for the RK4 integrator

```
function [ x ] = Rk4Integrator(obj, t, idx, x0 )
    function phi = phi(t, dt, x)
        k1_value = k1(obj.func, t, dt, 0       , x);
        k2_value = k2(obj.func, t, dt, k1_value, x);
        k3_value = k3(obj.func, t, dt, k2_value, x);
        k4_value = k4(obj.func, t, dt, k3_value, x);
        phi = (k1_value + 2*k2_value + 2*k3_value + k4_value)/6;
    end

    x = obj.Step(t, idx, x0, @phi);
end

function [k] = k1(f, t, ~, ~, x)
    k = f(t,x);
end
function [k] = k2(f, t, dt, k1, x)
    k = f(t+dt/2, x+dt*k1/2);
end
function [k] = k3(f, t, dt, k2, x)
    k = f(t+dt/2, x+dt*k2/2);
end
function [k] = k4(f, t, dt, k3, x)
    k = f(t+dt, x+dt*k3);
end
```

# A. Code tests

In order to test the MATLAB[3] code given in appendixA, a two test cases where considerd as given in the slides[1]. The fist considers the function $y = e^{2t}$ and its derivative $\dot{y} = 2 \cdot e^{2t}$. The iteration values are given in table 3. For the RK4 integrator, the k values are also given:

k1_value = 14.78

k2_value = 40.17

k3_value = 40.17

k4_value = 109.20

| time | Euler value | RK4 value |
|---|---|---|
| 1 | 7.39 | 7.39 |
| 2 | 22.17 | 54.83 |

**Table 3:** Test case 1: Results

For the second test case, the same set of equations as for the section 3. Only the start values differ from that section (a=7378.137 km, e=0.1, i=Ω=ω=M=0). The results of the Euler iterator are given in table 4 and for the RK4 integrator in table 5.

| | Vx | Vy | Vz | Rx | Ry | Rz |
|---|---|---|---|---|---|---|
| results.euler(t=0) | 0.00 | 8125.88 | 0.00 | 6640323.30 | 0.00 | 0.00 |
| results.euler(t=2) | -18.08 | 8125.88 | 0.00 | 6640323.30 | 16251.77 | 0.00 |
| results.euler(t=4) | -36.16 | 8125.84 | 0.00 | 6640287.14 | 32503.54 | 0.00 |

**Table 4:** Test case 2: Euler integrator

| | Vx | Vy | Vz | Rx | Ry | Rz |
|---|---|---|---|---|---|---|
| results.rk4(t=0) | 0.00 | 8125.88 | 0.00 | 6640323.30 | 0.00 | 0.00 |
| k1_value(t=0) | -9.04 | -0.00 | -0.00 | 0.00 | 8125.88 | 0.00 |
| k2_value(t=0) | -9.04 | -0.01 | -0.00 | -9.04 | 8125.88 | 0.00 |
| k3_value(t=0) | -9.04 | -0.01 | -0.00 | -9.04 | 8125.87 | 0.00 |
| k4_value(t=0) | -9.04 | -0.02 | -0.00 | -18.08 | 8125.86 | 0.00 |
| results.rk4(t=2) | -18.08 | 8125.86 | 0.00 | 6640305.22 | 16251.75 | 0.00 |
| k1_value(t=2) | -9.04 | -0.02 | -0.00 | -18.08 | 8125.86 | 0.00 |
| k2_value(t=2) | -9.04 | -0.03 | -0.00 | -27.12 | 8125.84 | 0.00 |
| k3_value(t=2) | -9.04 | -0.03 | -0.00 | -27.12 | 8125.83 | 0.00 |
| k4_value(t=2) | -9.04 | -0.04 | -0.00 | -36.16 | 8125.80 | 0.00 |
| results.rk4(t=4) | -36.16 | 8125.80 | 0.00 | 6640250.98 | 32503.42 | 0.00 |

**Table 5:** Test case 2: RK4 integrator