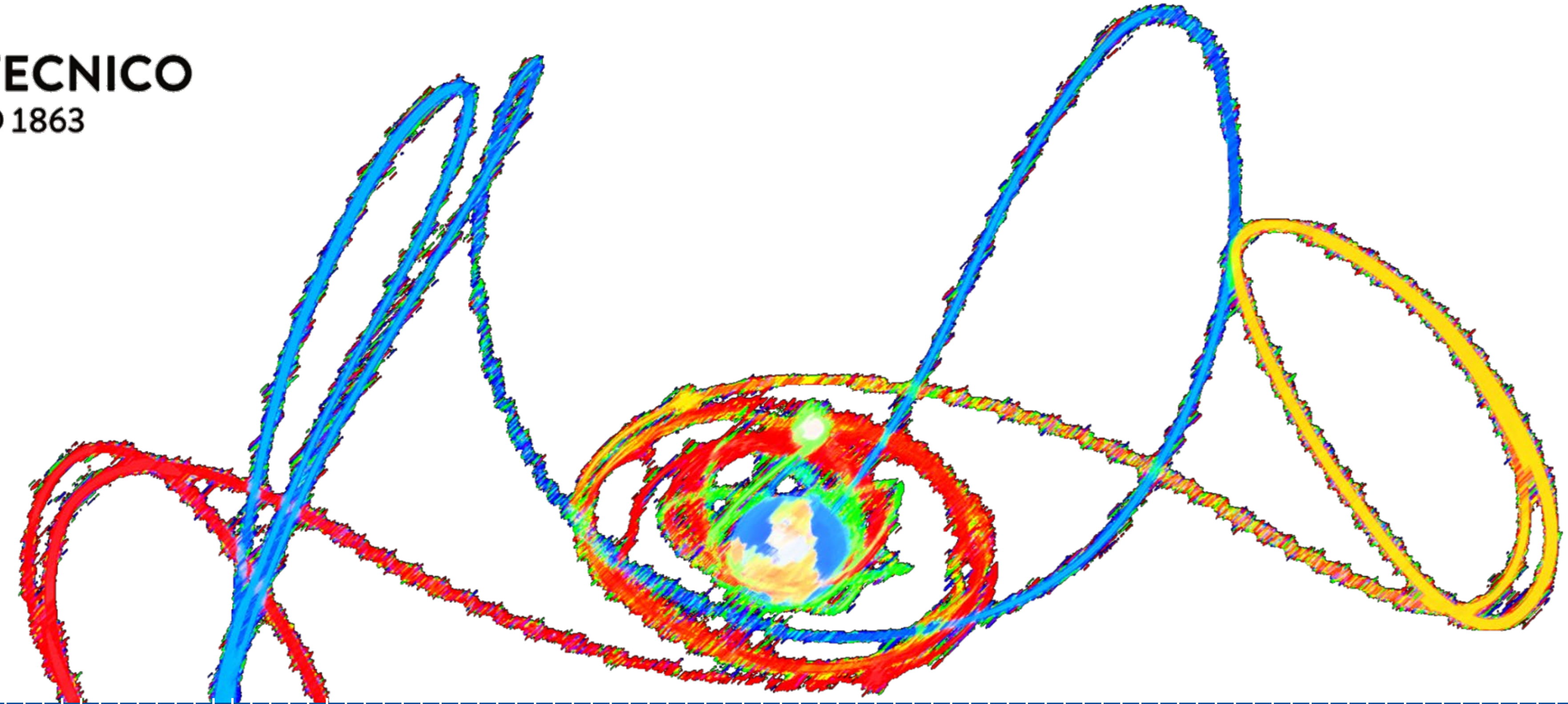


Orbital Mechanics

Lab Chapter 1: The two body problem

Andrea MUCIACCIA, Mathilda BOLIS, Francesca OTTOBONI, Giacomo BORELLI, Juan Luis GONZALO GOMEZ, Camilla COLOMBO

Department of Aerospace Science and Technology, Politecnico di Milano

Academic year 2025/26

Chapter Contents

Lab – The two body problem

POLITECNICO MILANO 1863

NUMERICAL INTEGRATION OF DYNAMICAL SYSTEMS

Ordinary Differential Equations

An **ordinary differential equation** (ODE) is an equation F involving one or more functions $\mathbf{x}$ of an independent variable t (usually **time**) and their derivatives:

$$F\left(t; \mathbf{x}(t), \dot{\mathbf{x}}(t), \ddot{\mathbf{x}}(t), \dots, \mathbf{x}^{(n)}(t)\right) = 0$$

Many dynamical systems in nature can be modelled through a system of ODEs

Example: Two body problem

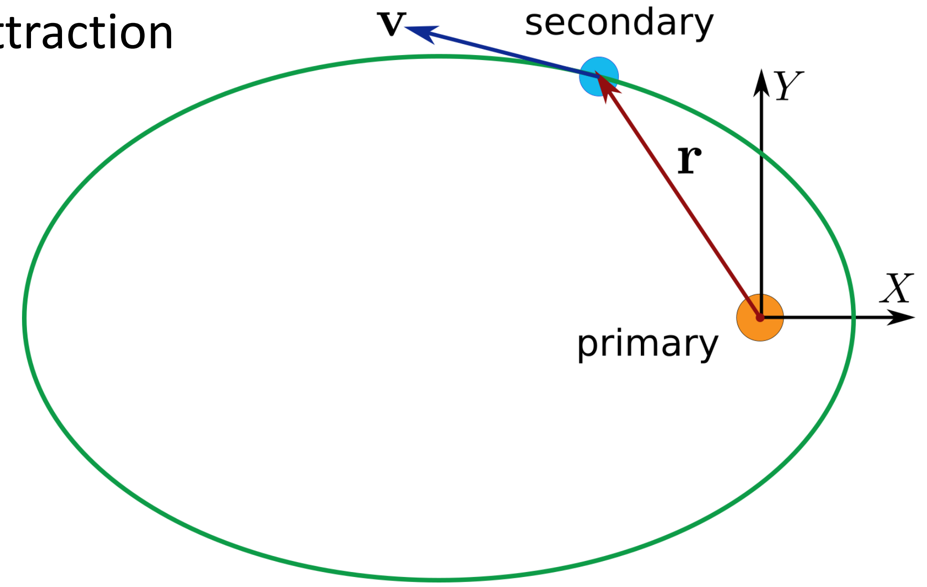
Motion of two bodies subjected only to their mutual gravitational attraction

Equations for the motion of the secondary body around the primary body:

$$\ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} = \mathbf{0}$$

where:

$\mathbf{r}$**:** Position vector of the secondary, in an inertial reference frame centred on the primary
μ: gravitational parameter of primary body



Reduction to a first-order system

The **order** of an ODE is that of the highest-order derivative involved.

Any ODE can be reduced to a **first-order system** by treating the derivatives up to order *n-1* as **independent variables**. In explicit form:

$$\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{dt}} = \mathbf{f}(t; \mathbf{y}(t))$$

where $\mathbf{y} = \left[\mathbf{x}, \dot{\mathbf{x}}, \ \dots, \mathbf{x}^{(n-1)}\right]$.

Example: Two body problem – reduction to a first-order ODE system

New state vector that includes the first derivative of the position (i.e., the velocity)

$$\mathbf{s} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

Velocity definition:

$$\dot{\mathbf{r}} = \mathbf{v}$$

Equations of motion:

$$\dot{\mathbf{v}} = -\frac{\mu}{r^3}\mathbf{r}$$

$$\dot{\mathbf{s}} = \begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -\frac{\mu}{r^3} r_x \\ -\frac{\mu}{r^3} r_y \\ -\frac{\mu}{r^3} r_z \end{bmatrix}$$

Resolution of an ODE system

If $\mathbf{f}$ is *regular enough*, the solution of the first order ODE system for given initial conditions $\mathbf{y}(t_0) = \mathbf{y}_0$ exists and is unique (Cauchy-Lipschitz or Picard-Lindelöf theorem)

- In some cases, a closed form (analytic) solution can be found
- In general, we will resort to **numerical integration schemes**

Example: Two body problem – analytic solution

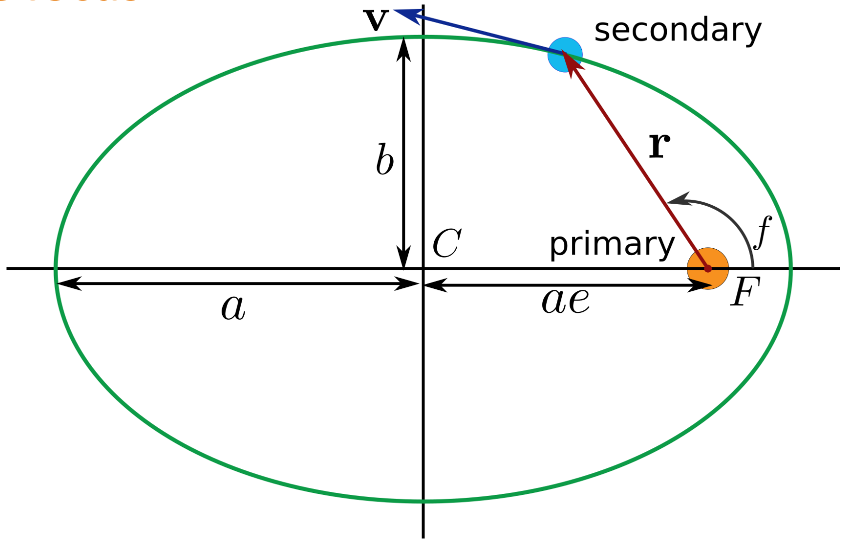
First Kepler's law: In an inertial frame centred on one of the bodies (**the primary**), the **other body** describes a **conic** (ellipse, hyperbola, or parabola) with the **primary in one focus**

Polar equation of the orbit:

$$r = \frac{p}{1 + e\cos f}$$

$\mathbf{r}$**:** Position vector of the secondary body
μ: gravitational parameter of primary body
p: semilatus rectum of the conic
e: eccentricity of the conic
f: true anomaly

F: focus
C: centre of the conic
a: semimajor axis
b: semiminor axis



Numerical resolution of ODEs

Euler scheme

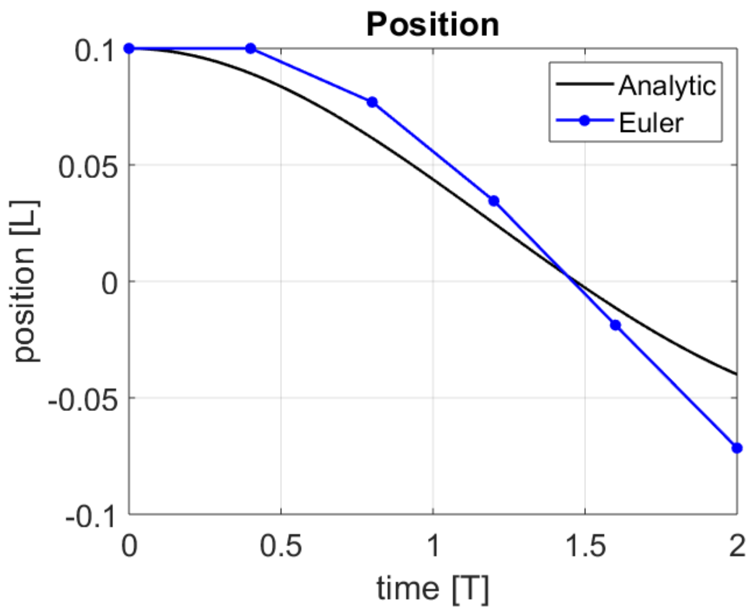
The simplest numerical scheme for solving ODEs is the **Euler scheme**. Consider the Taylor expansion of $\mathbf{y}(t)$ around t_k:

$$\mathbf{y}(t) = \mathbf{y}(t_k) + \dot{\mathbf{y}}(t_k)(t - t_k) + O(t - t_k)^2$$

Retaining only the first order term, **and recalling that $\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y})$**:

$$\mathbf{y}(t_{k+1}) = \mathbf{y}_{k+1} \approx \mathbf{y}_k + \mathbf{f}(t_k, \mathbf{y}_k)\Delta t$$

That is, we approximate the value of $\mathbf{y}$ at time step $t_{k+1} = t_k + \Delta t$ from its value at t_k. Geometrically, this is equivalent to treating $\mathbf{y}$ between t_k and t_{k+1} as a straight line with the slope given by $\mathbf{f}_k$.



The values of t_k and $\mathbf{y}_k$ are not known a priori, so $\mathbf{f}_k = \mathbf{f}(t_k, \mathbf{y}_k)$ has to be evaluated as the numerical solver advances

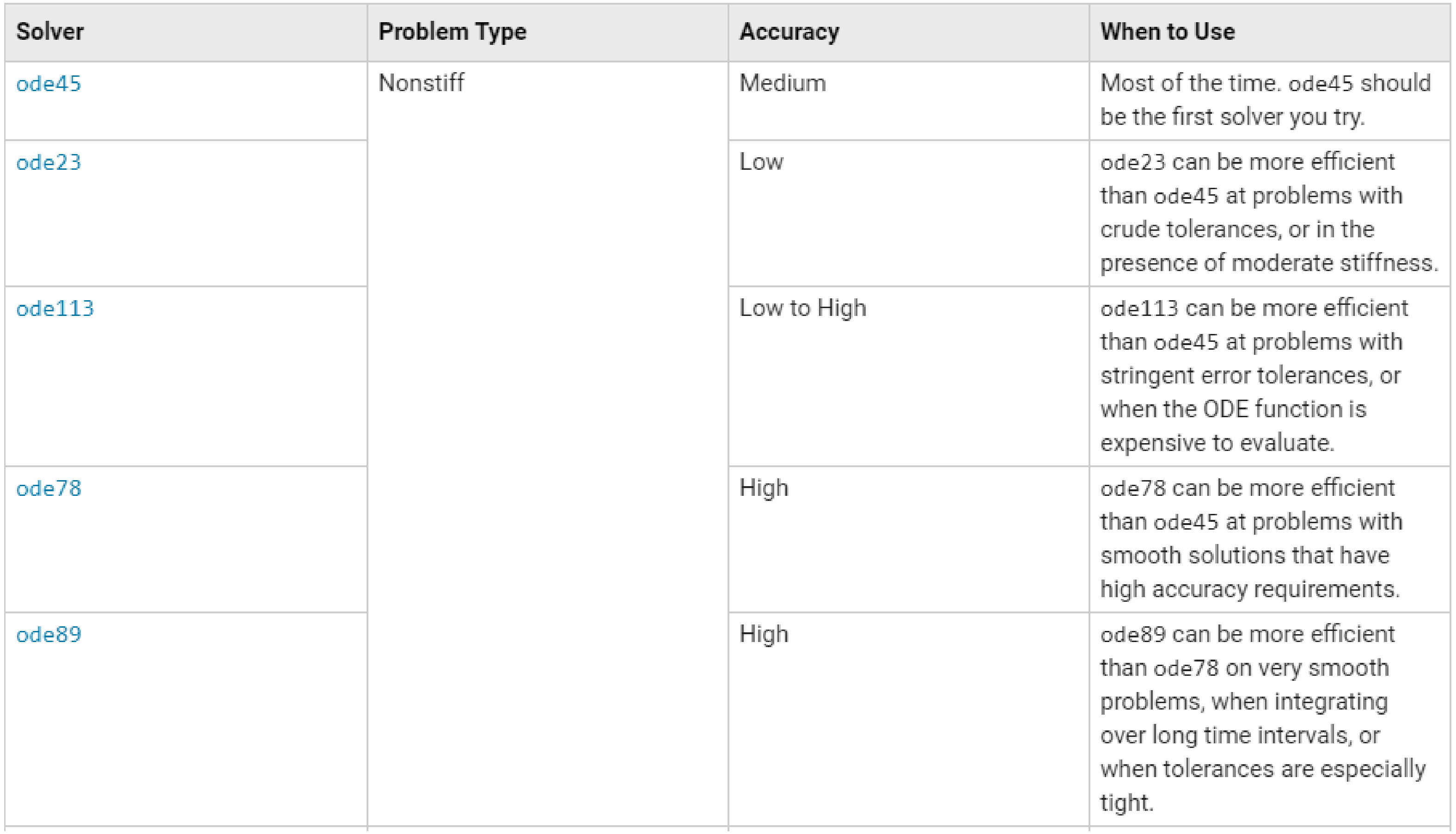
The Euler scheme requires very small time steps to achieve high precision (local error is of order $\ddot{\mathbf{y}}_k \Delta t^2$). This leads to large number of steps and computational time.

In practice, we will use more advanced integration schemes already implemented in MATLAB

Integrating ODEs with MATLAB

MATLAB provides several solvers for systems of ODEs

- Based on different integration schemes, with different properties
- **We have to select the most appropriate one for our dynamics**



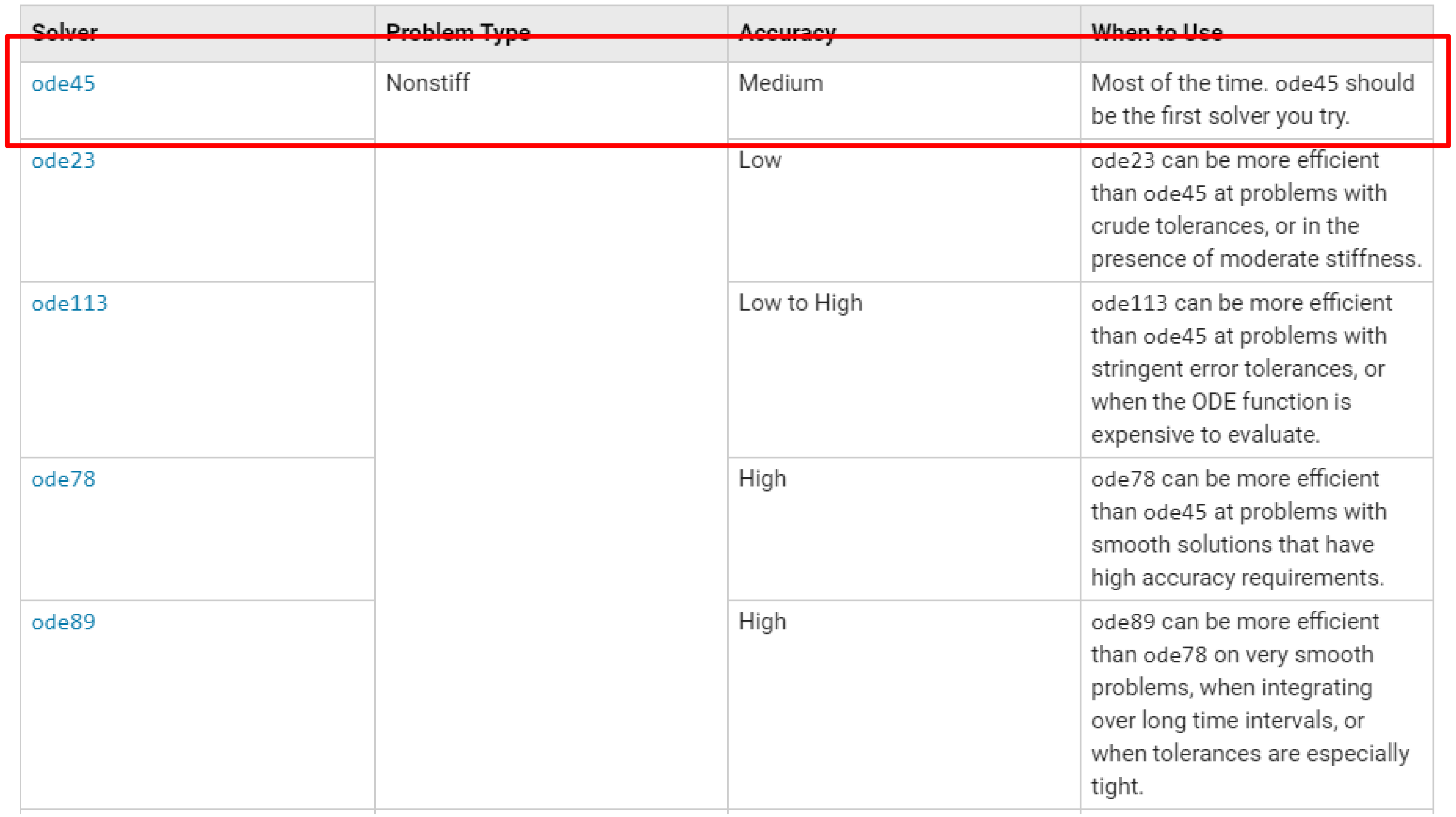
Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.

Integrating ODEs with MATLAB

ode45

MATLAB provides several solvers for systems of ODEs

- Based on different integration schemes, with different properties
- **We have to select the most appropriate one for our dynamics**



Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.

ode45
Explicit Runge-Kutta (4,5) formula (the Dormand-Prince pair)

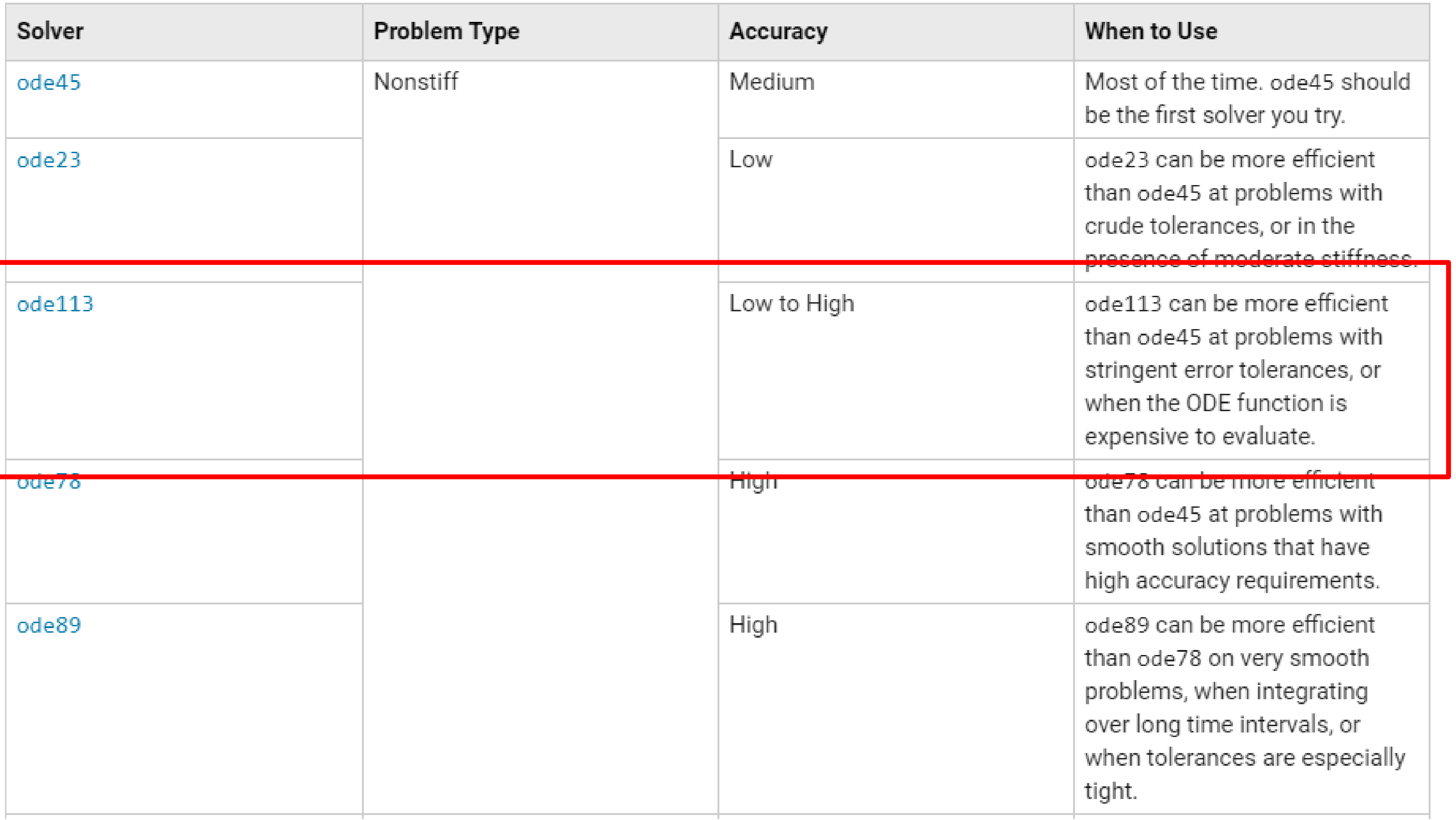
- Very versatile
- Low performance for stiff problems
- Low performance for high accuracy requirements

Integrating ODEs with MATLAB

ode113

MATLAB provides several solvers for systems of ODEs

- Based on different integration schemes, with different properties
- **We have to select the most appropriate one for our dynamics**



Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.

ode113

Variable-step, variable-order Adams-Bashforth-Moulton Predictor-Corrector solver of orders 1 to 13

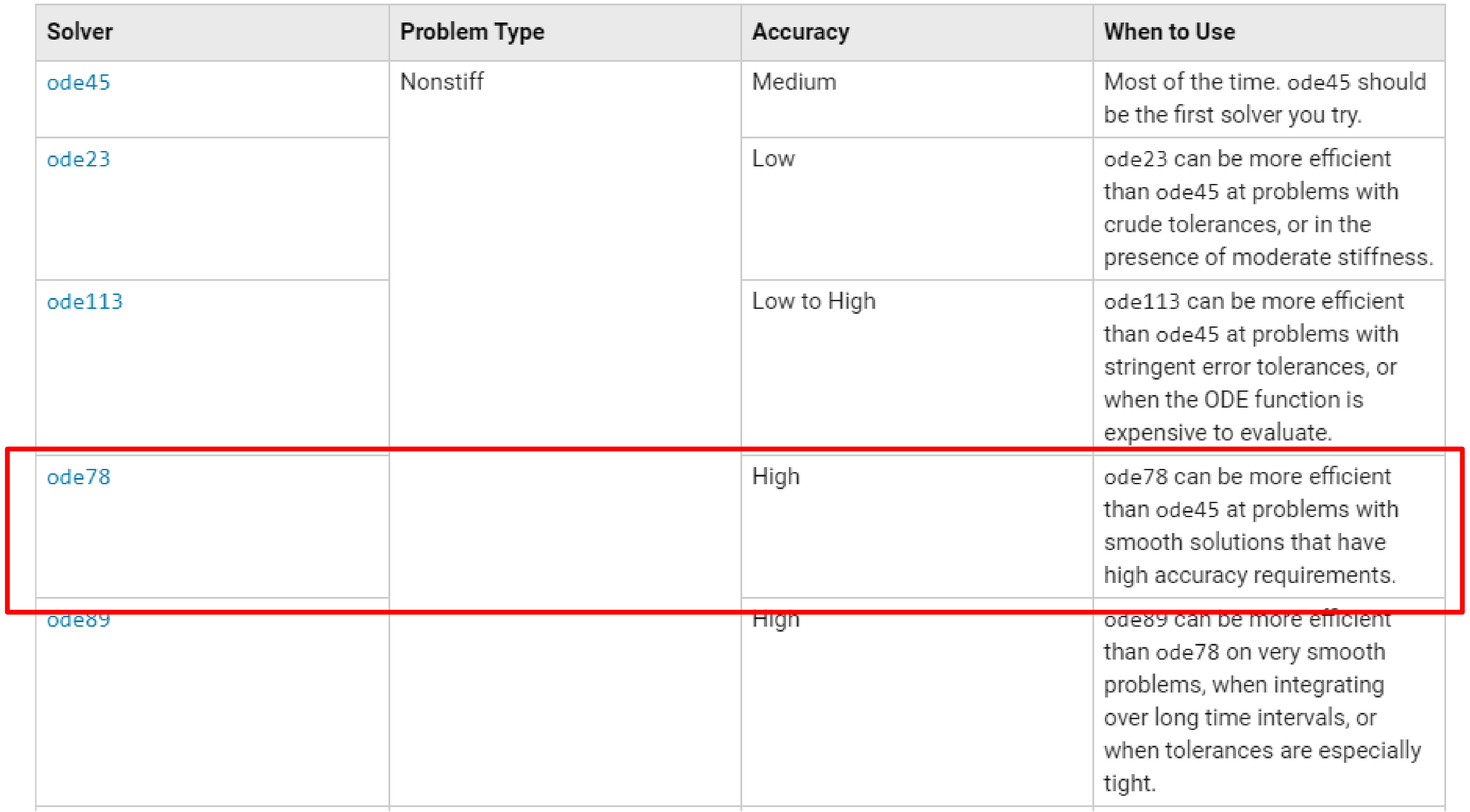
- Less function calls than ode45 (less evaluations of the ODE function)

Integrating ODEs with MATLAB

ode78

MATLAB provides several solvers for systems of ODEs

- Based on different integration schemes, with different properties
- **We have to select the most appropriate one for our dynamics**



Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.

ode78

Explicit Runge-Kutta (7,8) formula of high order (Verner's pair)

- More efficient (less computational cost) than ode45 for smooth problems (e.g., weakly perturbed orbits)
- Less versatile than ode45
- Low performance for stiff problems

Syntax for ODE solvers

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

Syntax for ODE solvers

Inputs: ODE function

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

`odefun` is a MATLAB function for the **right-hand side of the first order ODE system**, that is, function $\mathbf{f}$ in:

$$\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{d}t} = \mathbf{f}(t;\mathbf{y}(t))$$

This function has to be of the form

```
function dy = odefun( t, y )
    .....
    .....
end
```

⚠

`t` is a scalar

`y` and `dy` are column vectors of dimension [n x 1]

The order of inputs t and y must be respected

How do we introduce additional parameters (such as μ in the two-body problem)?

Syntax for ODE solvers

Inputs: ODE function

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

Two ways of introducing additional parameters:

1. Additional inputs added at the end of the argument list of `odeXX` (i.e., after `options`) are passed directly to `odefun`:

```
[t, y] = odeXX(@ odefun, tspan, y0, options, par1, par2, … )

              dy = odefun( t, y, par1, par2, … )
```

2. Use an **anonymous function**:

```
[t, y] = odeXX(@(t,y) odefun(t,y,par1,par2), tspan, y0, options )
```

This creates an unnamed function with inputs `t` and `y`, to be used by `odeXX` as a wrapper for `odefun`.

Syntax for ODE solvers

Inputs: Time span

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

Time span for the integration. There are two possibilities:

1. **`tspan = [ tstart  tend  ]`**
   If `tspan` is a 2-elements array, they represent the initial and final times for the integration, respectively. Output is given at the **internal time steps used by the solver (which depend on the tolerances required).**

2. **`tspan = [ tstart  t1  t2  …  tend  ]`**
   If `tspan` is a monotonic array of more than 2 elements, the solver returns the value of `y` only at the times in `tspan`. The first and last elements of `tspan` still represent the initial and final time of the integration. **This does not affect the internal time steps automatically decided by the solver (it uses its own time steps to integrate, and afterwards interpolates to get the values for the requested times).**

⚠ The time values in `tspan` must be strictly monotonic!

Syntax for ODE solvers

Inputs: Initial conditions

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

Column array with the initial conditions for the state (i.e., the value of `y` at the first time given in `tspan`, $\mathbf{y}(t_{\text{start}}) = \mathbf{y}_0$). It must have the same dimensions as `y` and `dy`.

Syntax for ODE solvers

Inputs: Options

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

Object containing optional parameters for the ODE solver. It is created using the `odeset` function (check the documentation center). For example:

```
options = odeset( 'RelTol', 1e-13, 'AbsTol', 1e-14 );
```

For now, we consider only the relative and absolute tolerances (`RelTol` and `AbsTol`, respectively). **The internal time steps used by the solver are automatically chosen to fulfill the tolerances (they do not depend on `tspan`). The number of internal steps only depends on the tolerances (and the problem being solved).**

The call to `odeset` only creates the variable containing the options. Do not forget to pass this variable to `odeXX` as the fourth input, for the options to be applied

There are many (powerful) options that can be configured. Check the **documentation center page** about `odeset` for more information

Default tolerance values, `RelTol=1e-3` and `AbsTol=1e-6`, are too loose for orbit propagation. Don't forget to set more stringent ones!

Syntax for ODE solvers

Outputs: Two possibilities

```
[t, y] = odeXX( odefun, tspan, y0, options )
```

`t`: m x 1 array with the times at each of the m time steps.

`y`: m x n array with the n states at each of the m time steps. That is, row m corresponds to the state at the m-th time step.

```
sol = odeXX( odefun, tspan, y0, options )
```

`sol`: MATLAB structure containing detailed information about the solution:

- Time and state at the time steps decided by the integrator are stored in `sol.x` and `sol.y`, respectively. **Beware**, they are transposed with respect to `t` and `y`: `sol.x` is a 1 x m array, and `sol.y` is a n x m array with each column corresponding to a different time step.
- The state for other times can be obtained using the function `deval`.

Back to the two-body problem

ODE function

```
function dy = ode_2bp( ~, y, mu )
%ode_2bp ODE system for the two-body problem (Keplerian motion)
%
% PROTOTYPE
%   dy = ode_2bp( t, y, mu )
%
% INPUT:
%   t[1]        Time (can be omitted, as the system is autonomous) [T]
%   y[6x1]      State of the body ( rx, ry, rz, vx, vy, vz ) [ L, L/T ]
%   mu[1]       Gravitational parameter of the primary [L^3/T^2]
%
% OUTPUT:
%   dy[6x1]     Derivative of the state  [ L/T^2, L/T^3 ]
%
% CONTRIBUTORS:
%   Juan Luis Gonzalo Gomez
%
% VERSIONS
%   2018-09-26: First version
%
% ------------------------------------------------------------------------

% Position and velocity
r = y(1:3);
v = y(4:6);

% Distance from the primary
rnorm = norm(r);

% Set the derivatives of the state
dy = [  v
        (-mu/rnorm^3)*r ];

end
```

Back to the two-body problem

ODE function

```
function dy = ode_2bp( ~ y, mu )
%ode_2bp ODE system for the two-body problem (Keplerian motion)
%
% PROTOTYPE
%   dy = ode_2bp( t, y, mu )
%
% INPUT:
%   t[1]        Time (can be omitted, as the system is autonomous) [T]
%   y[6x1]      State of the body ( rx, ry, rz, vx, vy, vz ) [ L, L/T ]
%   mu[1]       Gravitational parameter of the primary [L^3/T^2]
%
% OUTPUT:
%   dy[6x1]     Derivative of the state  [ L/T^2, L/T^3 ]
%
% CONTRIBUTORS:
%   Juan Luis Gonzalo Gomez
%
% VERSIONS
%   2018-09-26: First version
%
% -------------------------------------------------------------------------

% Position and velocity
r = y(1:3);
v = y(4:6);

% Distance from the primary
rnorm = norm(r);

% Set the derivatives of the state
dy = [  v
        (-mu/rnorm^3)*r ];

end
```

Use ~ to omit unneeded inputs (like time in the two-body problem)

Always document your code!
You can take this example as template.

Back to the two-body problem

Main script

```
% Physical parameters
mu_E = astroConstants(13); % Earth's gravitational parameter [km^3/s^2]

% Initial condition
r0 = [ 26578.137; 0; 0 ]; % [km]
v0 = [ 0; 2.221; 3.173 ]; % [km/s]
y0 = [ r0; v0 ];

% Set time span
a = 1/( 2/norm(r0) - dot(v0,v0)/mu_E ); % Semi-major axis [km]
T = 2*pi*sqrt( a^3/mu_E ); % Orbital period [s]
tspan = linspace( 0, 2*T, 1000 );

% Set options for the ODE solver
options = odeset( 'RelTol', 1e-13, 'AbsTol', 1e-14 );
% Perform the integration
[ T, Y ] = ode113( @(t,y) ode_2bp(t,y,mu_E), tspan, y0, options );

% Plot the results
figure()
plot3( Y(:,1), Y(:,2), Y(:,3), '-' )
xlabel('X [km]'); ylabel('Y [km]'); zlabel('Z [km]');
title('Two-body problem orbit');
axis equal;
grid on;
```

Exercise 1: Two-body problem

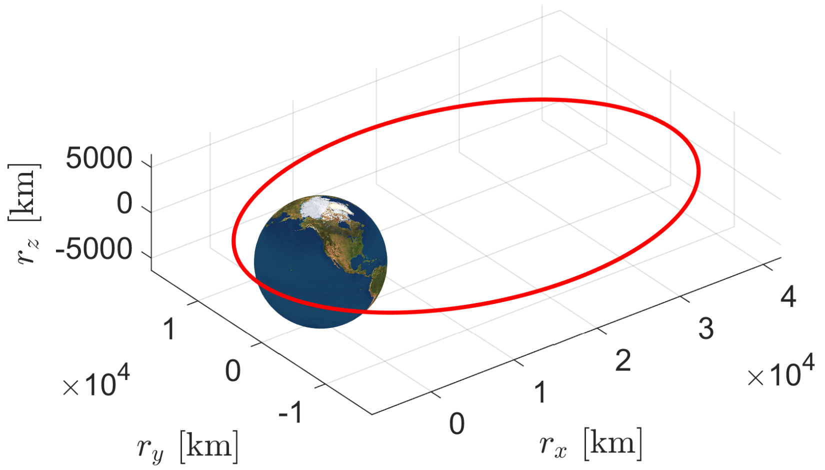
Exercise 1: Integrate numerically a Keplerian orbit (two-body problem)

1. Implement the code to propagate the orbit[1]:
   - Identify the **states** of the system and the **physical parameters** involved
   - Write the *second-order ODE* describing dynamics
   - Reduce the problem to a *first-order ODE system*
   - Implement the `odefun` MATLAB **function** for this ODE system
   - Write a **main script** to integrate numerically the system, **choosing one of MATLAB's solvers and setting its options as needed**.

Equations of motion:

$$\ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} = \mathbf{0}$$

μ: gravitational parameter of primary body



[1] **Orbit propagation**: prediction of the orbital characteristics of a body at some future date given the current orbital characteristics.

Exercise 1: Two-body problem

Exercise 1: Integrate numerically a Keplerian orbit (two-body problem)

2. Propagate the orbit for different initial conditions $(\mathbf{r}_0, \mathbf{v}_0)$ and primary attracting body:
   - You can get the gravitational parameter μ of different bodies in the solar system from function `astroConstants.m` in WeBeep.
   - To have a closed orbit (elliptical orbit), your initial conditions should correspond to an energy $\varepsilon < 0$
3. Analyse the results:
   - Plot the orbit over 1 period T
     - Only elliptical (i.e., closed) orbits have a period. Hyperbolic and parabolic (i.e., open) orbits can also be propagated, but they will never close.
   - Plot the components and norm of angular momentum vector $\mathbf{h}$ and eccentricity vector $\mathbf{e}$ over 5 periods, and check that they remain constant in magnitude and direction
   - Check that $\mathbf{h}$ and $\mathbf{e}$ remain perpendicular (hint: plot the error of a suitable test condition)
   - Plot the specific energy ε over 5 periods, and check if it is constant in time
   - Plot the evolution of the radial and transversal components of the velocity over 5 periods

Exercise 1: Two-body problem

Useful relations

Orbital period	$T = 2\pi\sqrt{a^3/\mu}$	Specific energy	$\varepsilon = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a}$
Angular momentum	$\mathbf{h} = \mathbf{r} \times \mathbf{v}$	Eccentricity vector	$\mathbf{e} = \frac{1}{\mu}\mathbf{v} \times \mathbf{h} - \frac{\mathbf{r}}{r}$
Semilatus rectum	$p = a(1 - e^2) = \frac{h^2}{\mu}$		

Radial unit vector $\mathbf{u}_r = \frac{\mathbf{r}}{r}$

Out-of-plane unit vector $\mathbf{u}_h = \frac{\mathbf{h}}{h}$

Transversal unit vector $\mathbf{u}_t = \mathbf{u}_h \times \mathbf{u}_r$

Radial velocity $v_r = \mathbf{v} \cdot \mathbf{u}_r$

Transversal velocity $v_t = \mathbf{v} \cdot \mathbf{u}_t$

Exercise 1: Two-body problem

Useful MATLAB functions

- Some MATLAB functions that are useful for computing these quantities:
  - `dot`: Computes the dot product (scalar product) of two vectors. If the inputs are two matrices, it treats them as a collection of column vectors and computes the result for each pair of them. Check the **Documentation Center** for more information and how to choose a different dimension to operate
  - `cross`: Computes the cross product (vectorial product) of two vectors. Same as `dot`, it can work with collections of vectors given in matrix form
  - `norm`: Computes the norm of a vector or matrix. Several norms are supported, by default it computes the 2-norm (the usual Euclidean norm)
  - `vecnorm`: Computes the norm of multiple vectors, given as column vectors of a matrix

- Other useful function for future labs is `atan2(y,x)`. It computes the arctangent of `y/x`, determining the correct angular quadrant automatically based on the signs of `y` and `x`
  - Instead, remember that inverse trigonometric functions `asin`, `acos`, and `atan` give results in an interval of length π, and there will be another valid solution in the full 2π interval. You must make sure that you are choosing the correct one

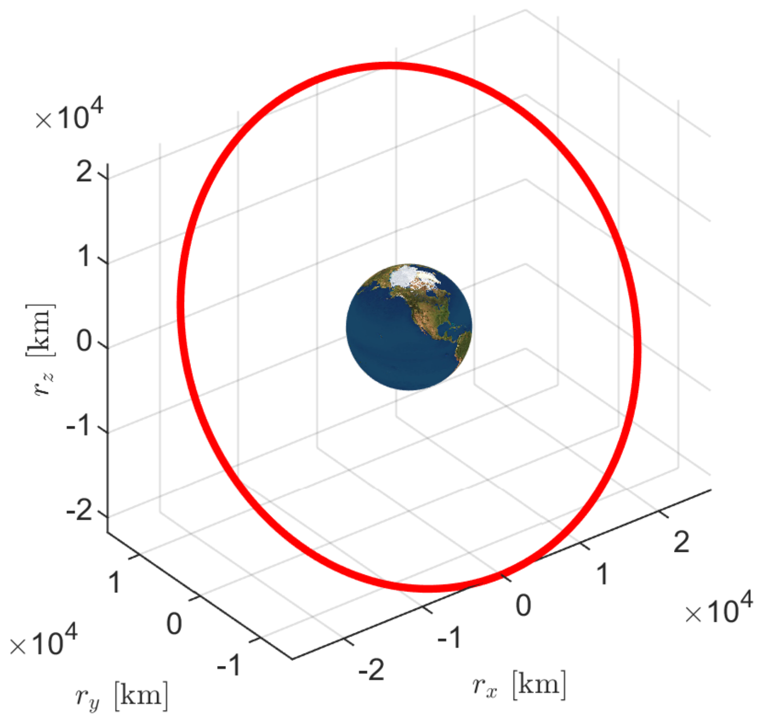
Exercise 1: Two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

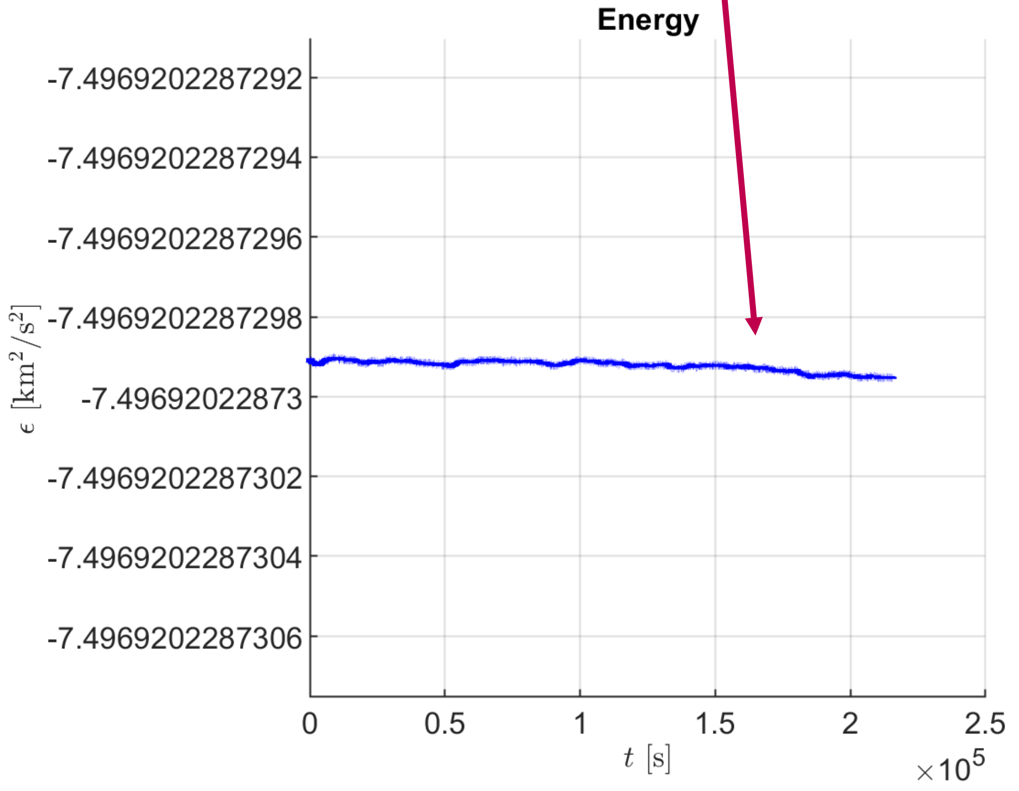
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$
$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$

These small variations are numerical error. They depend on the way you programmed your code, your Matlab version, and hardware, so they will not be exactly equal (but should be similarly small).



Orbit representation



Specific energy

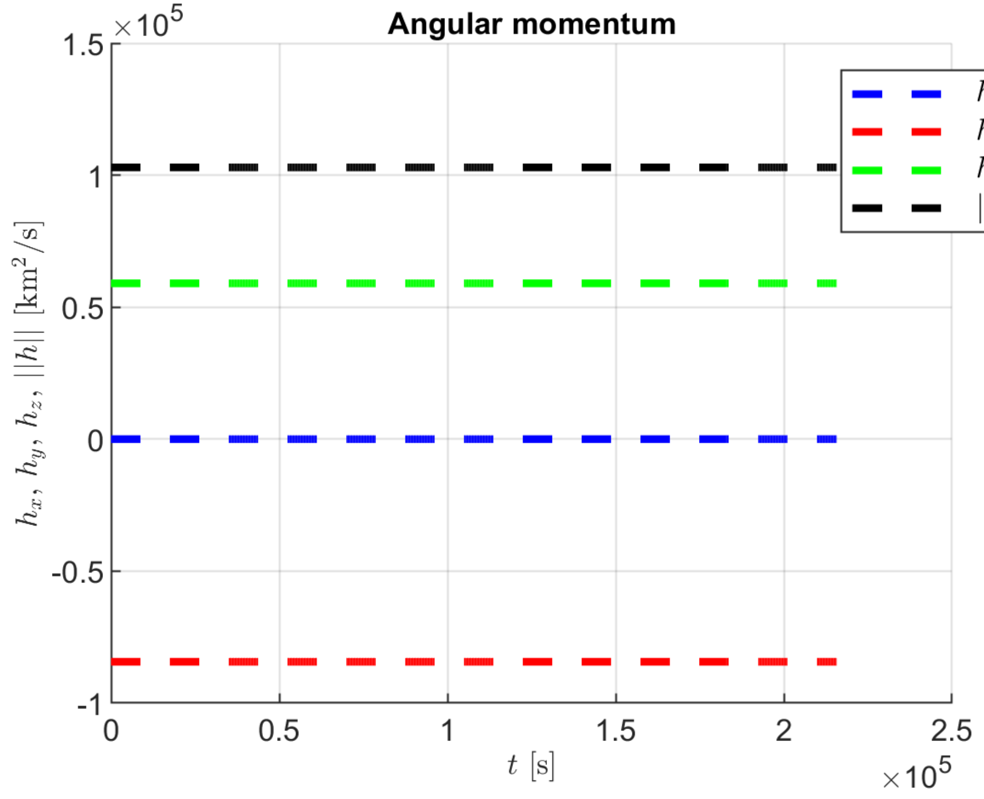
Exercise 1: Two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

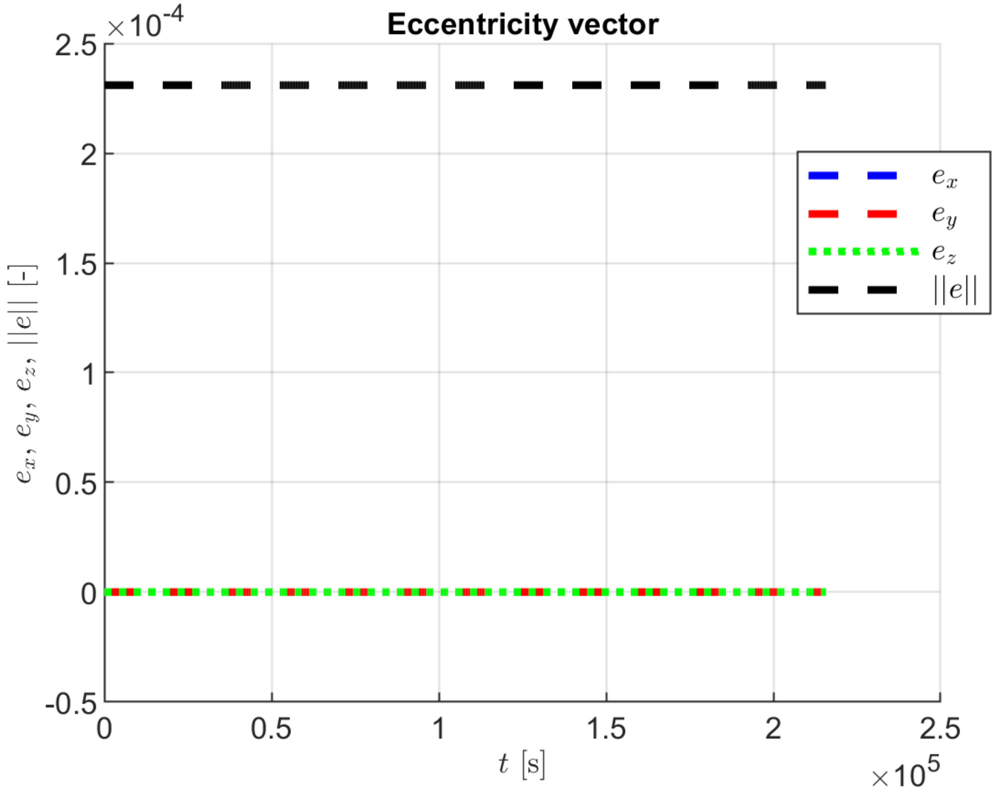
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$

$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Angular momentum



Eccentricity vector

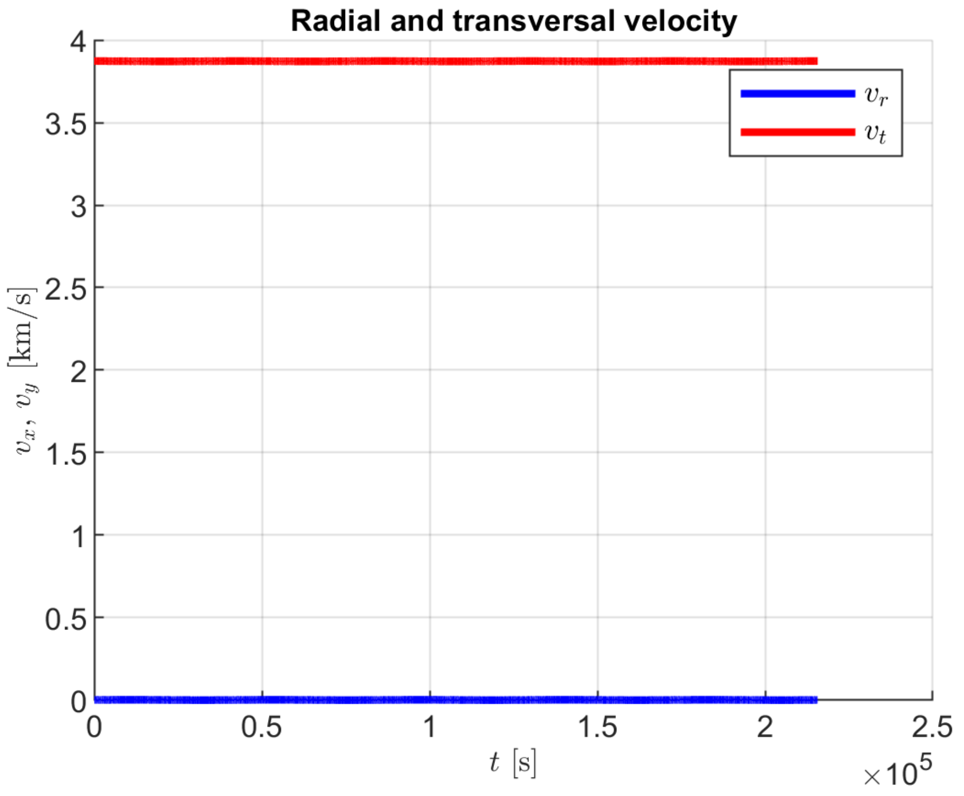
Exercise 1: Two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

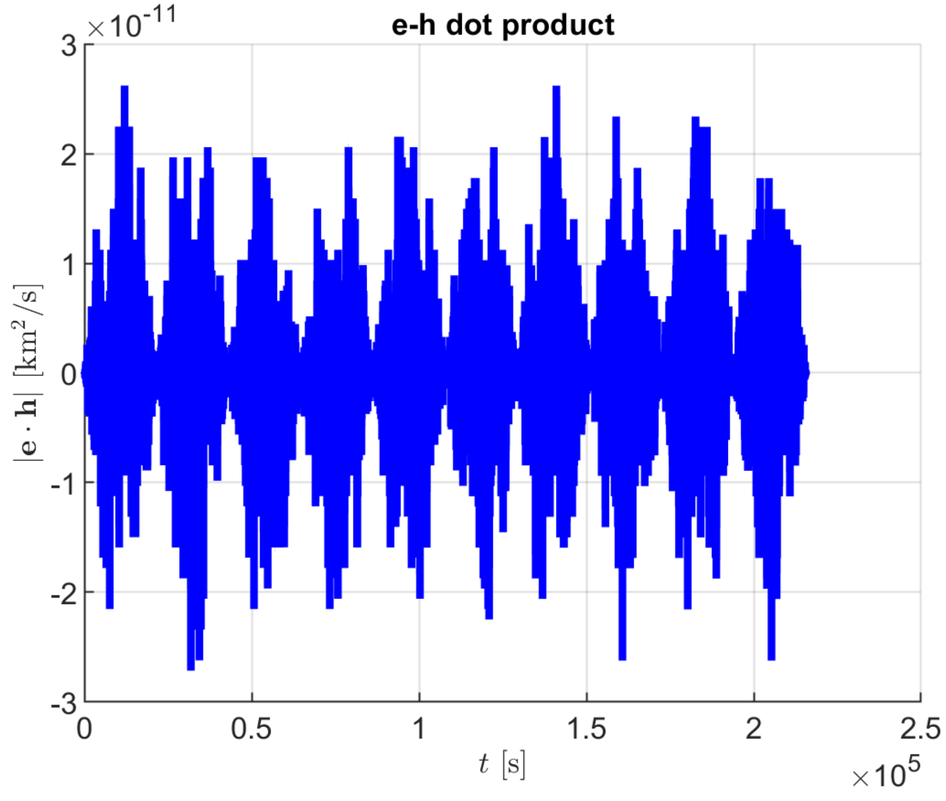
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$

$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Radial and transversal velocity



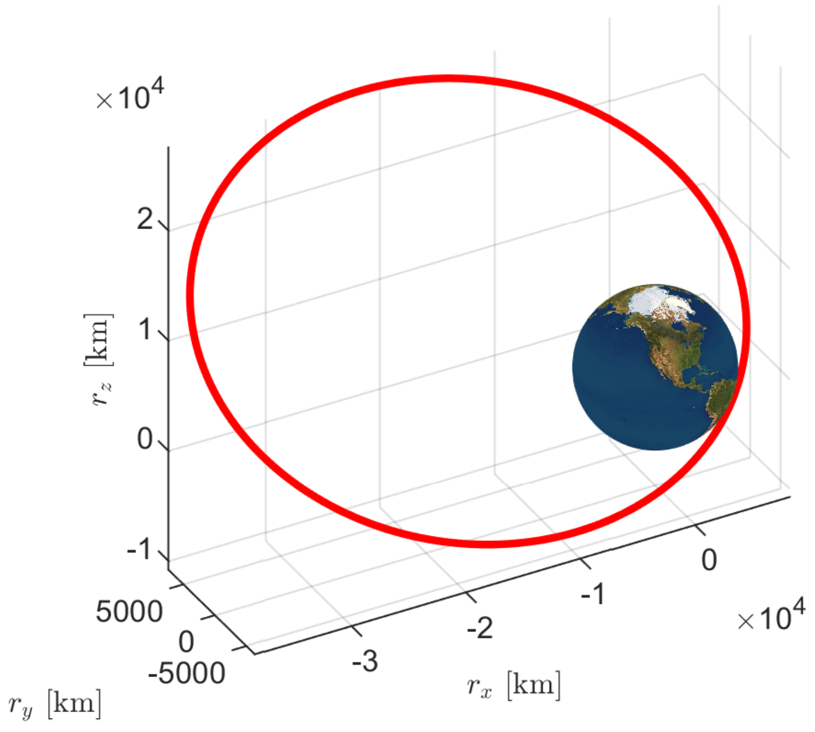
Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$

Exercise 1: Two-body problem

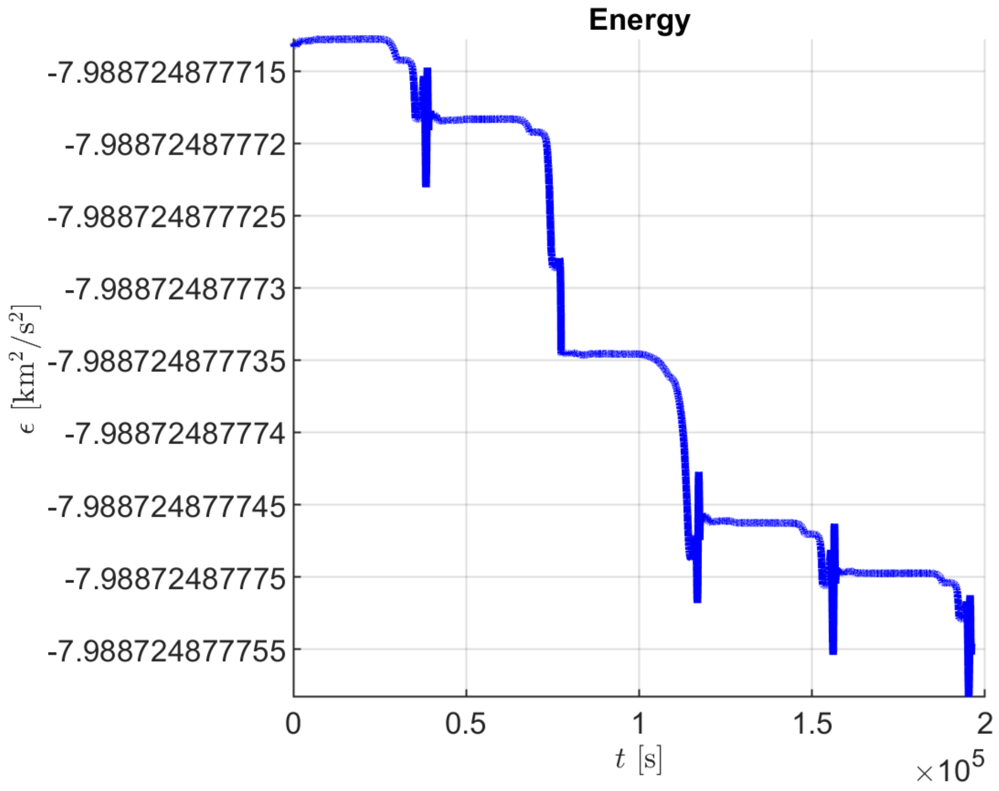
Sample solution: Highly eccentric and inclined orbit

Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$
$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Orbit representation



Specific energy

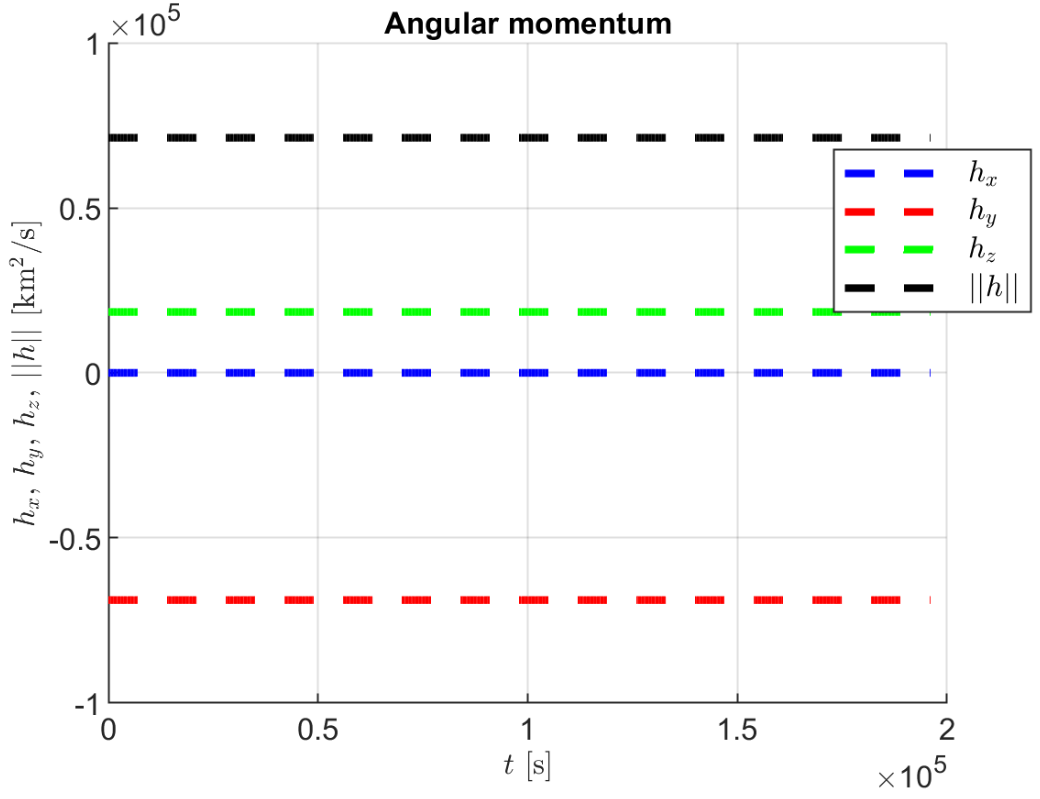
Exercise 1: Two-body problem

Sample solution: Highly eccentric and inclined orbit

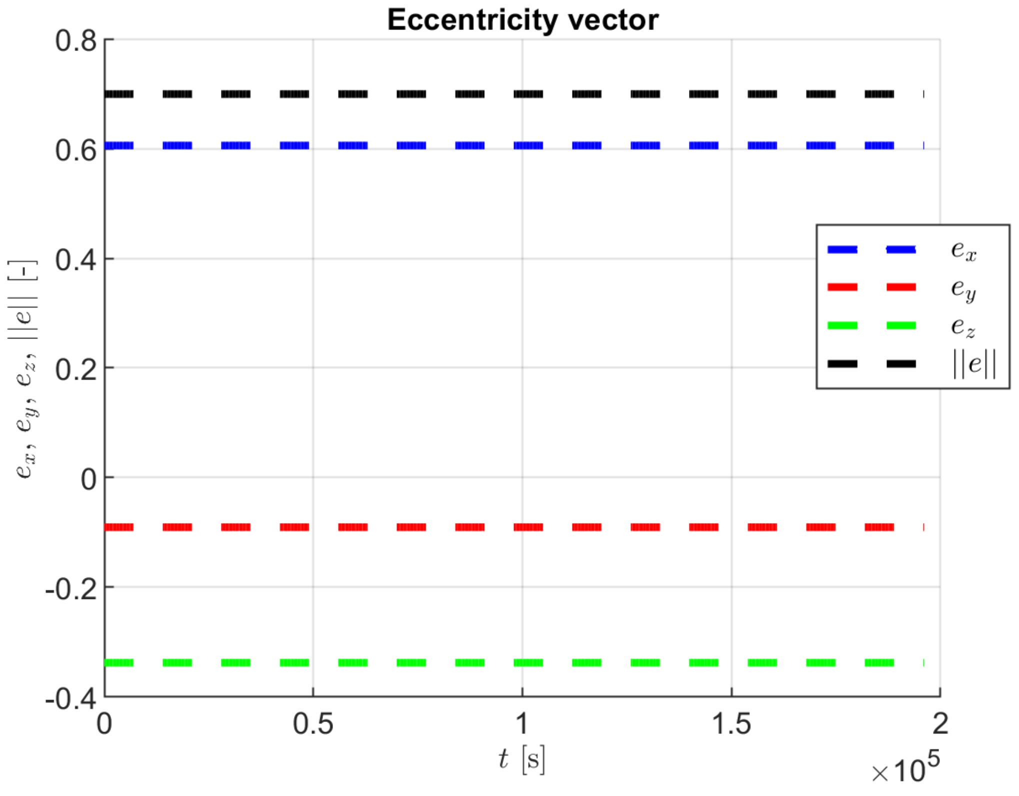
Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$

$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Angular momentum



Eccentricity vector

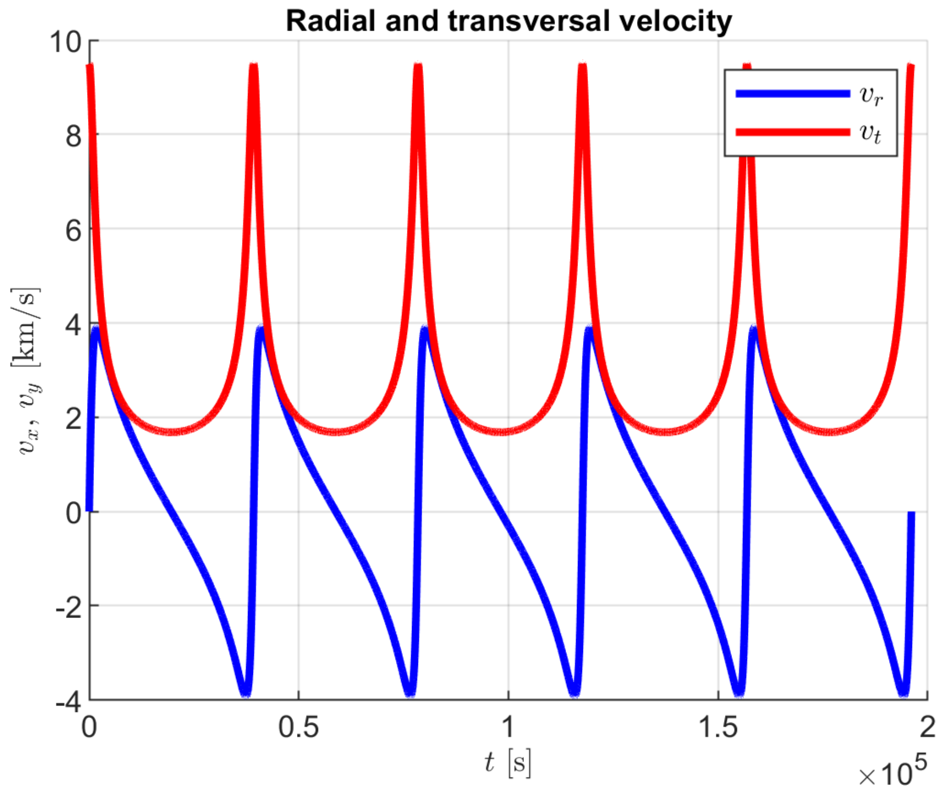
Exercise 1: Two-body problem

Sample solution: Highly eccentric and inclined orbit

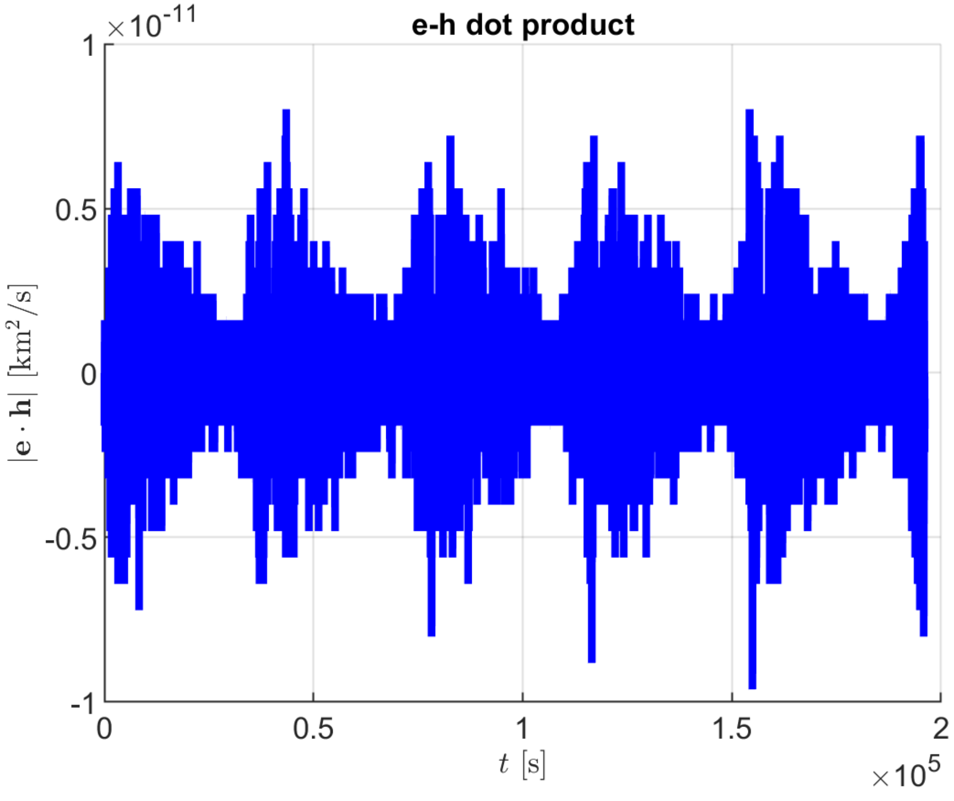
Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622] \text{ km}$$

$$\mathbf{v}_0 = [4.752, 2.130, 7.950] \text{ km/s}$$



Radial and transversal velocity



Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$

Perturbed two-body problem

Non-spherical gravity field and other perturbations

- The two-body problem only accounts for **2 bodies with spherical gravity fields:**
  - In real life, the two bodies **will not be homogeneous spheres**
  - Many other perturbations will be present, depending on the orbital region: atmospheric drag, solar radiation pressure, gravitational attraction from other bodies, etc. We will study them in detail in Chapter 5
  - Spacecraft may perform manoeuvres using their propulsion systems

- Non-sphericity of Earth:
  - The Earth bulges out at the equator due to centrifugal forces, taking the form of an **oblate spheroid**
  - **Zonal variations:** the oblateness causes the gravitational field to depend not only on distance, but also on **latitude**. These **zonal variations** of the gravitational field can be expressed mathematically as a **series**, being the major contribution the **second zonal harmonic J_2**

Perturbed two-body problem

Earth’s oblateness – Effect of J_2

In Cartesian formulation ($\mathbf{r}$ and $\mathbf{v}$ in inertial reference frame), the perturbation due to the **second zonal harmonic J_2** can be expressed as:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_{J_2}$$

$$\mathbf{a}_{J_2} = \frac{3}{2}\frac{J_2 \mu R_e^2}{r^4}\left[\frac{x}{r}\left(5\frac{z^2}{r^2} - 1\right)\hat{\mathbf{i}} + \frac{y}{r}\left(5\frac{z^2}{r^2} - 1\right)\hat{\mathbf{j}} + \frac{z}{r}\left(5\frac{z^2}{r^2} - 3\right)\hat{\mathbf{k}}\right]$$

where:

- R_e is Earth’s equatorial radius
- $J_2 = 0.00108263$ is a constant coefficient (value taken from [1])
- $\mathbf{r} = x\,\hat{\mathbf{i}} + y\,\hat{\mathbf{j}} + z\,\hat{\mathbf{k}}$

[1] https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

Exercise 2: Perturbed two-body problem

Effect of J_2

Exercise 2: Earth orbit propagation with J_2

1. Modify the function from **Exercise 1** to include also the effect of J_2
   - Add physical parameters J_2 and R_e as inputs to the function
   - Remember that the value of $\mathbf{a}_{J_2}$ has to be recomputed at each time step

2. Repeat the analysis in point **3.** of **Exercise 1**, for initial conditions corresponding to Earth-bound elliptical orbits
   - Plot together and compare the results with and without J_2
   - Are all the conservation principles for the 2BP still valid for the orbit propagation including J_2?
   - Propagate for a 1-year timespan and check how the differences between both models accumulate in time

Data

$\mu_\oplus$, R_e, and J_2 from `astroConstants.m` (identifiers `13`, `23`, and `9`, respectively)

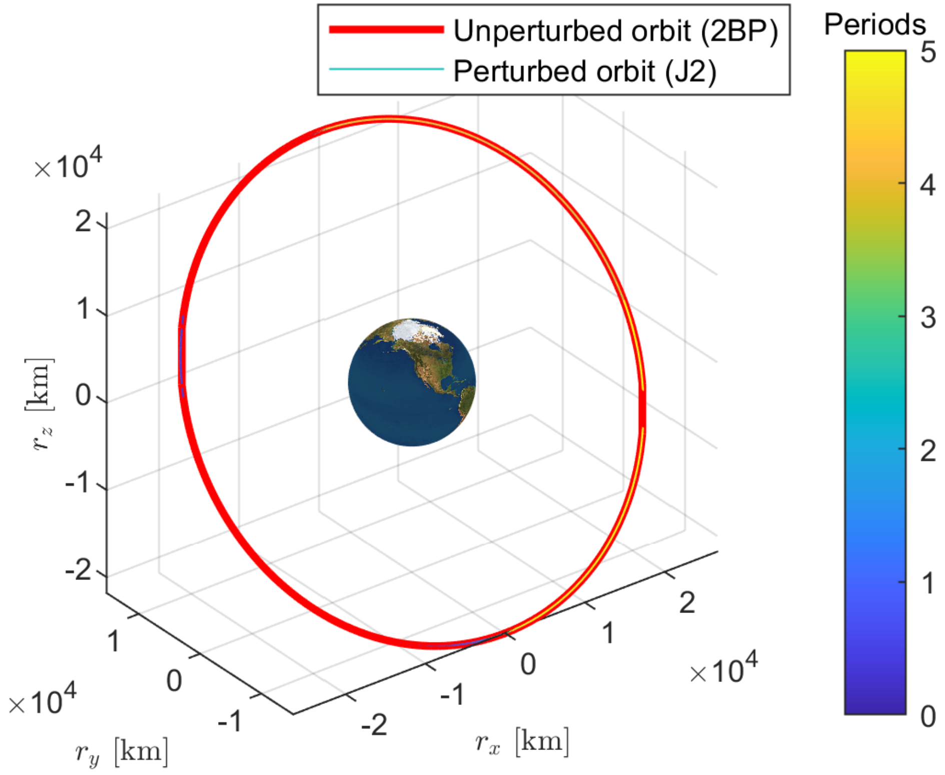
Exercise 2: Perturbed two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

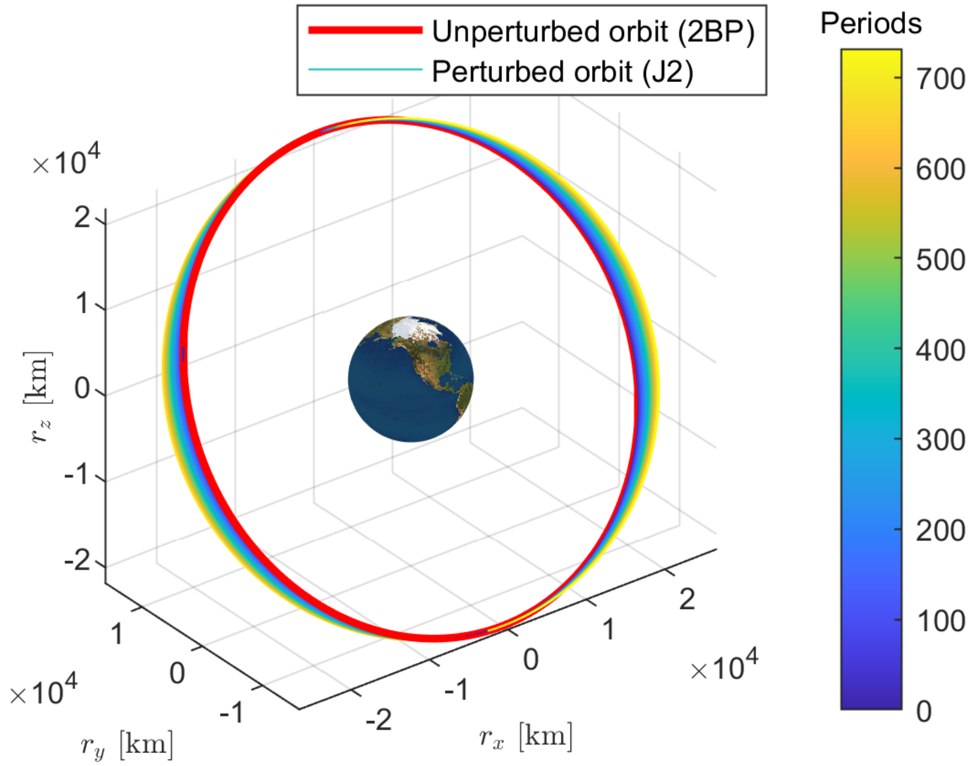
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$

$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Orbit representation – 5 periods



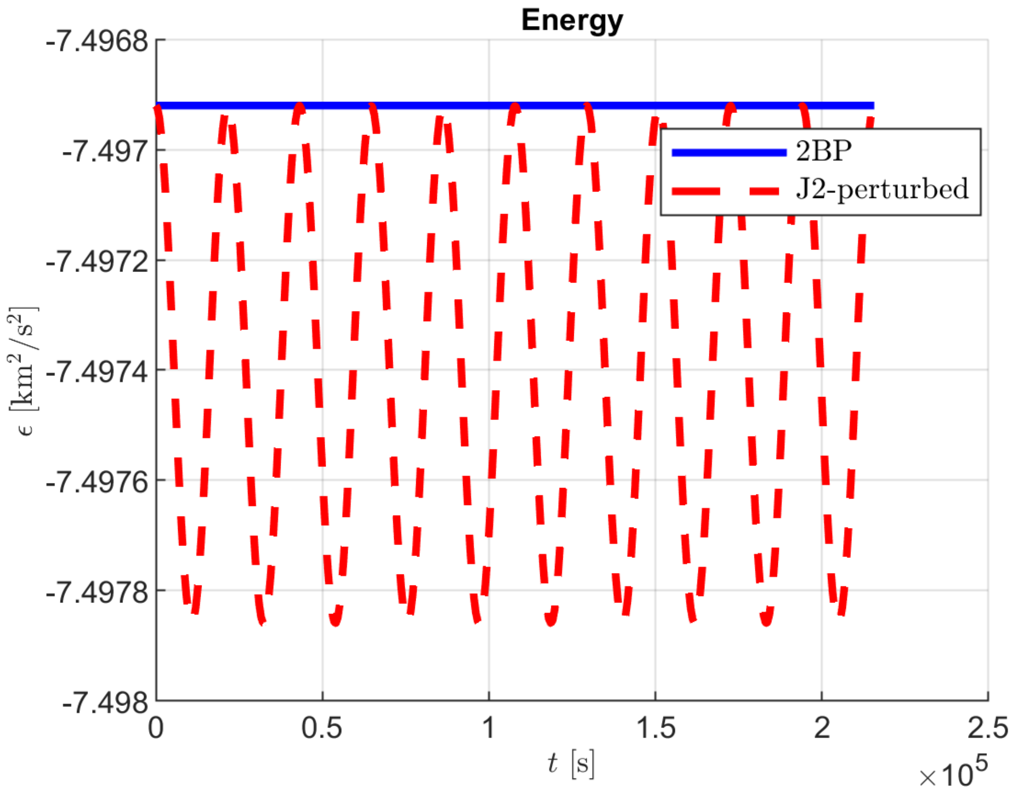
Orbit representation – 1 year

Exercise 2: Perturbed two-body problem

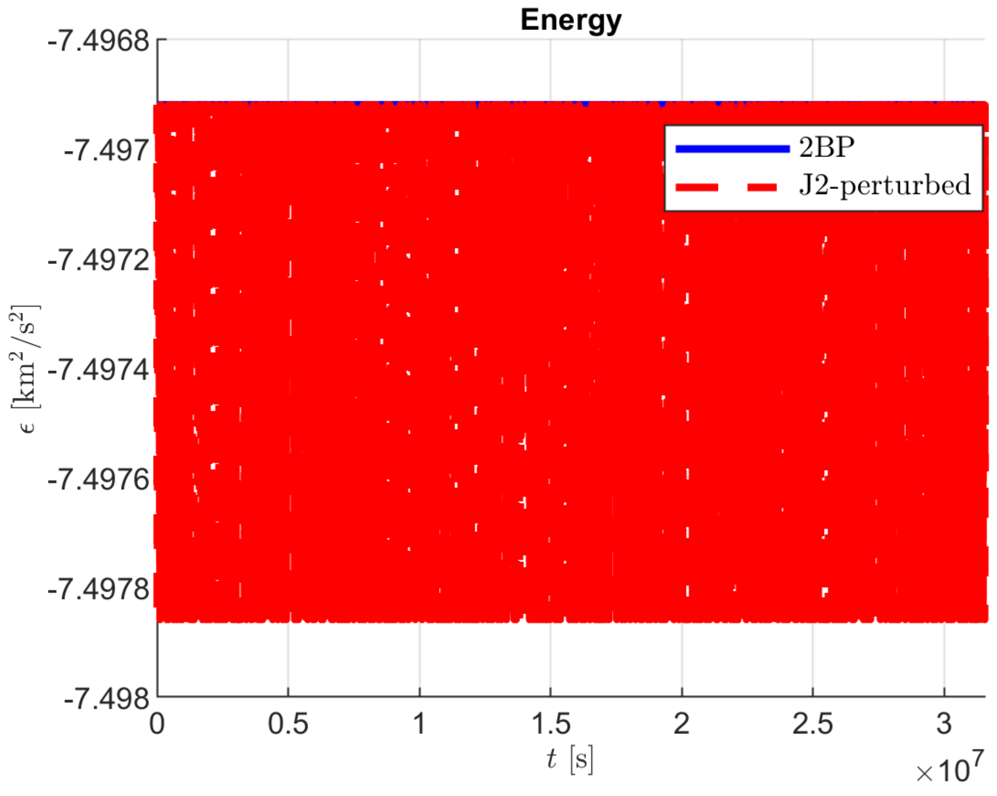
Sample solution: Quasi-circular Medium Earth Orbit

Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$
$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Specific energy – 5 periods



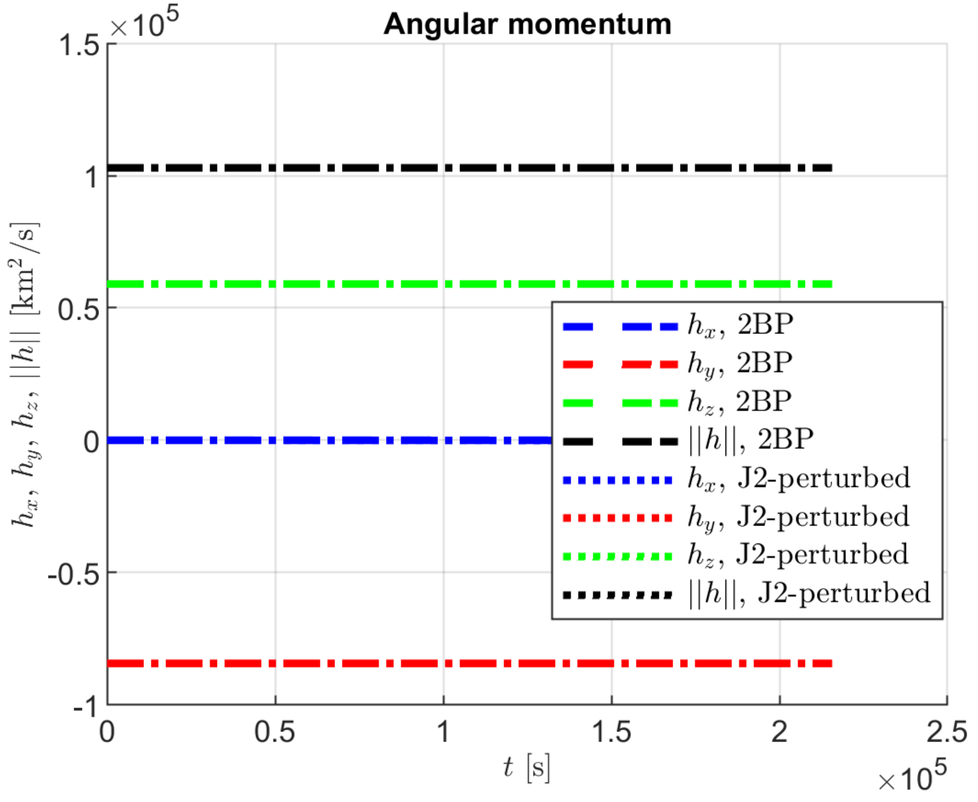
Specific energy – 1 year

Exercise 2: Perturbed two-body problem

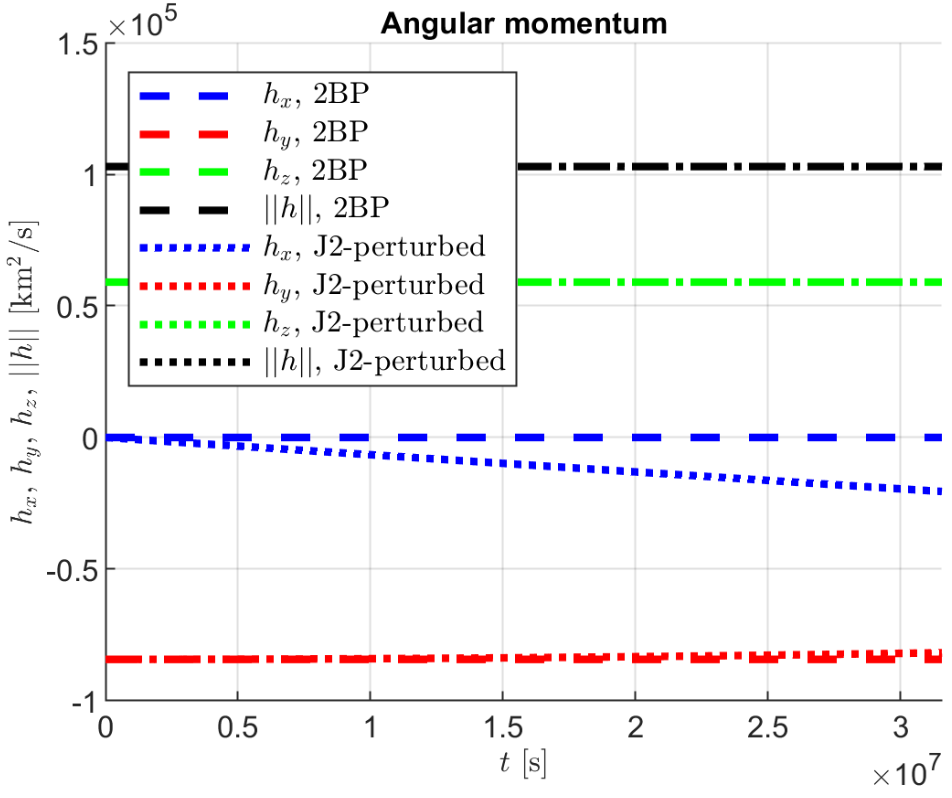
Sample solution: Quasi-circular Medium Earth Orbit

Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$
$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Angular momentum – 5 periods



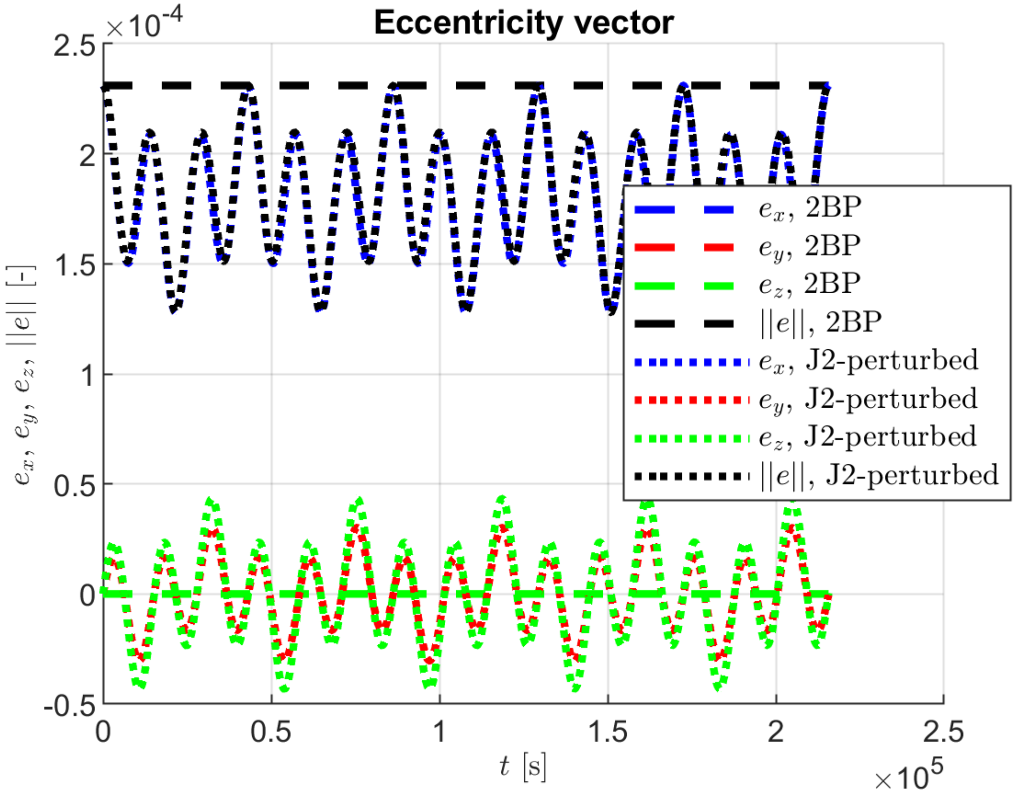
Angular momentum – 1 year

Exercise 2: Perturbed two-body problem

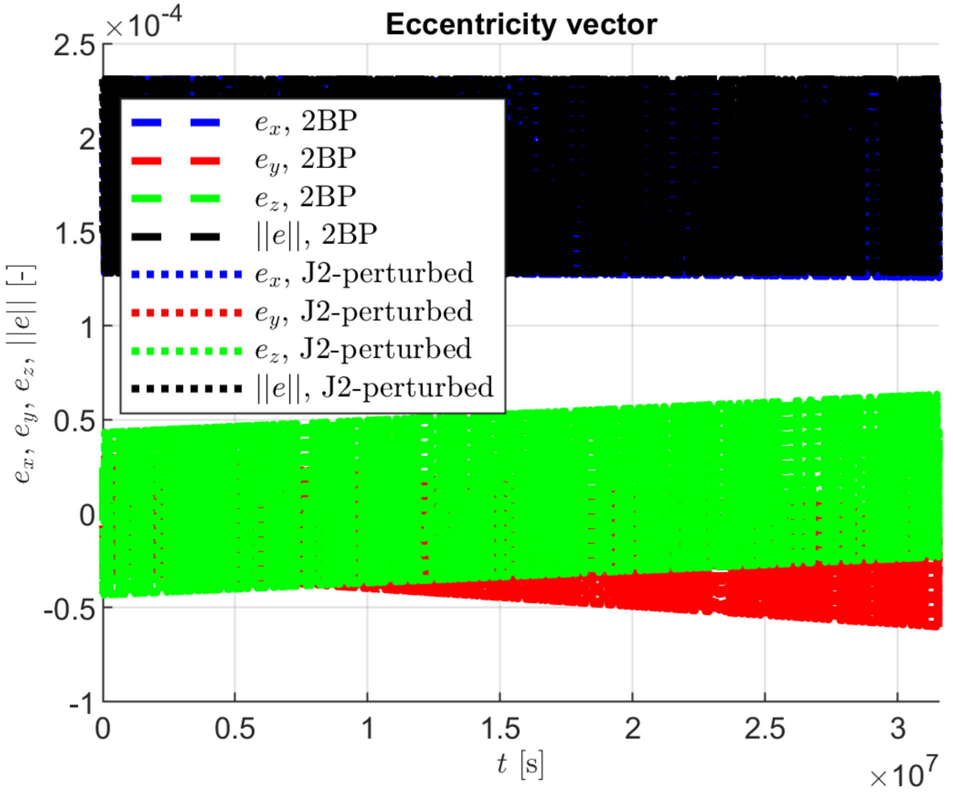
Sample solution: Quasi-circular Medium Earth Orbit

Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$
$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Eccentricity vector – 5 periods



Eccentricity vector – 1 year

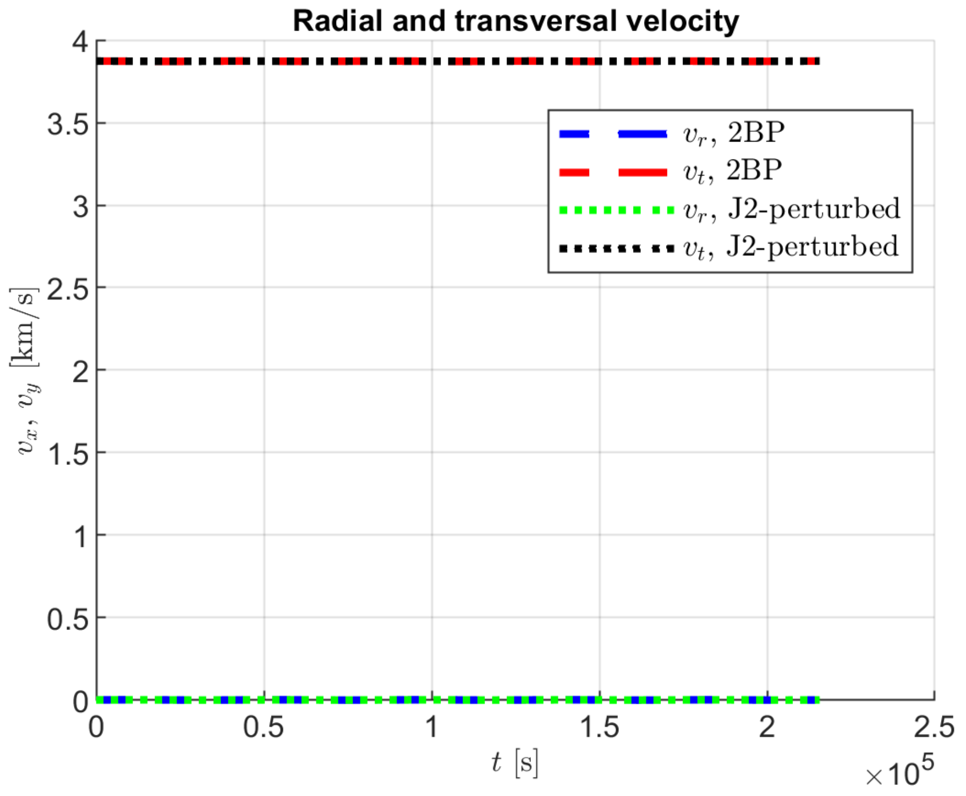
Exercise 2: Perturbed two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

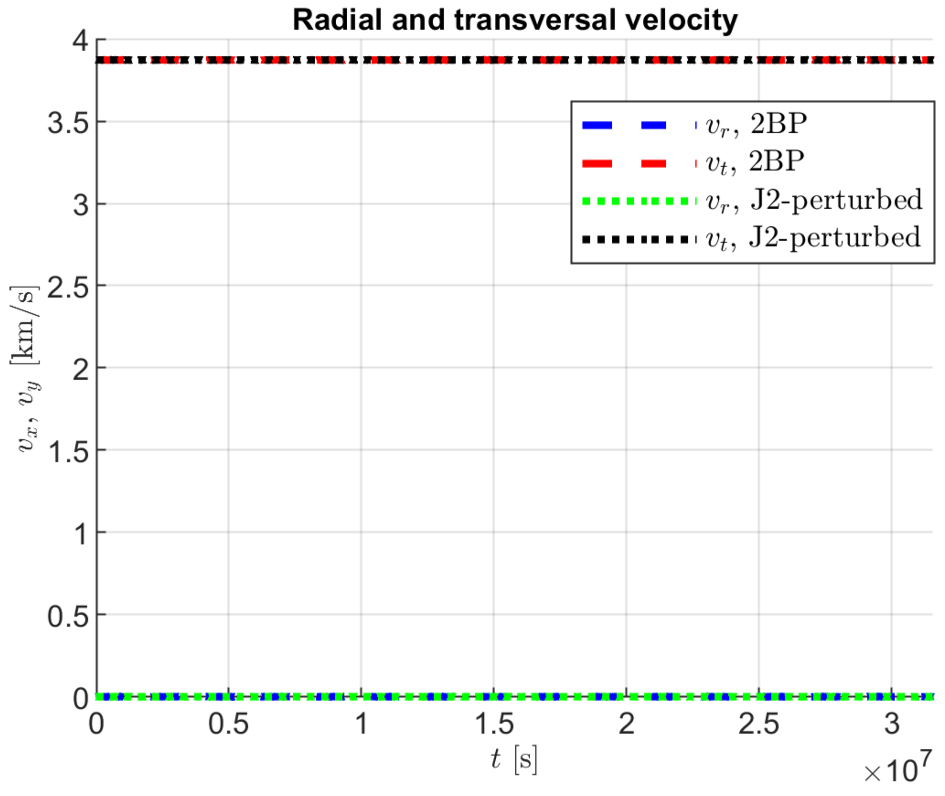
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$

$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Radial and transversal velocity – 5 periods



Radial and transversal velocity – 1 year

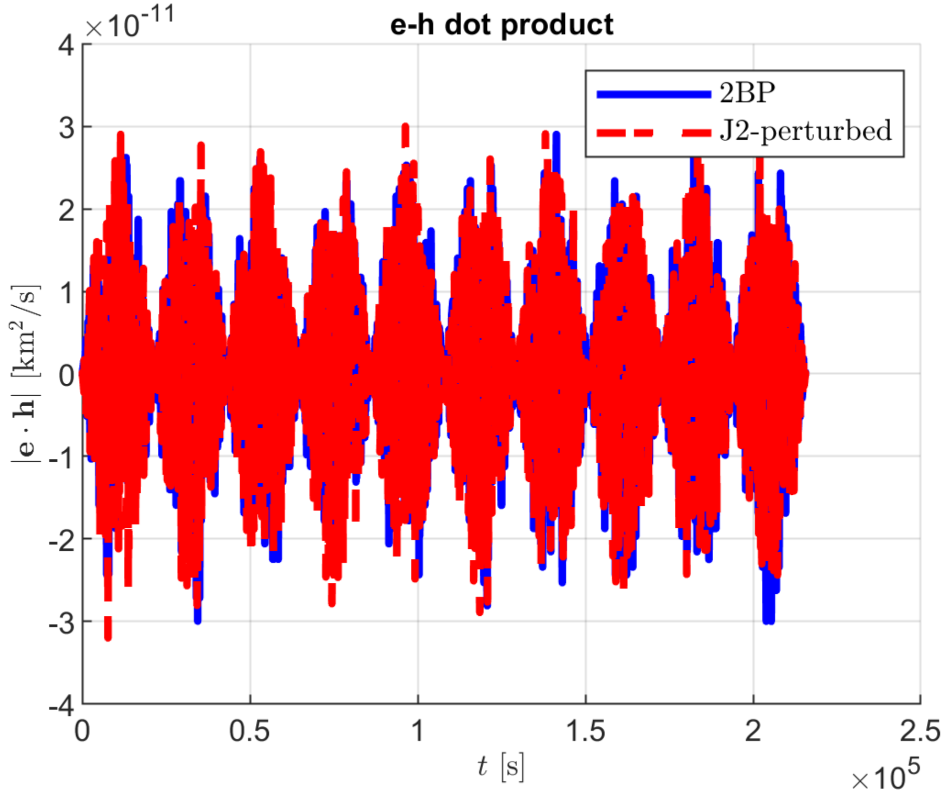
Exercise 2: Perturbed two-body problem

Sample solution: Quasi-circular Medium Earth Orbit

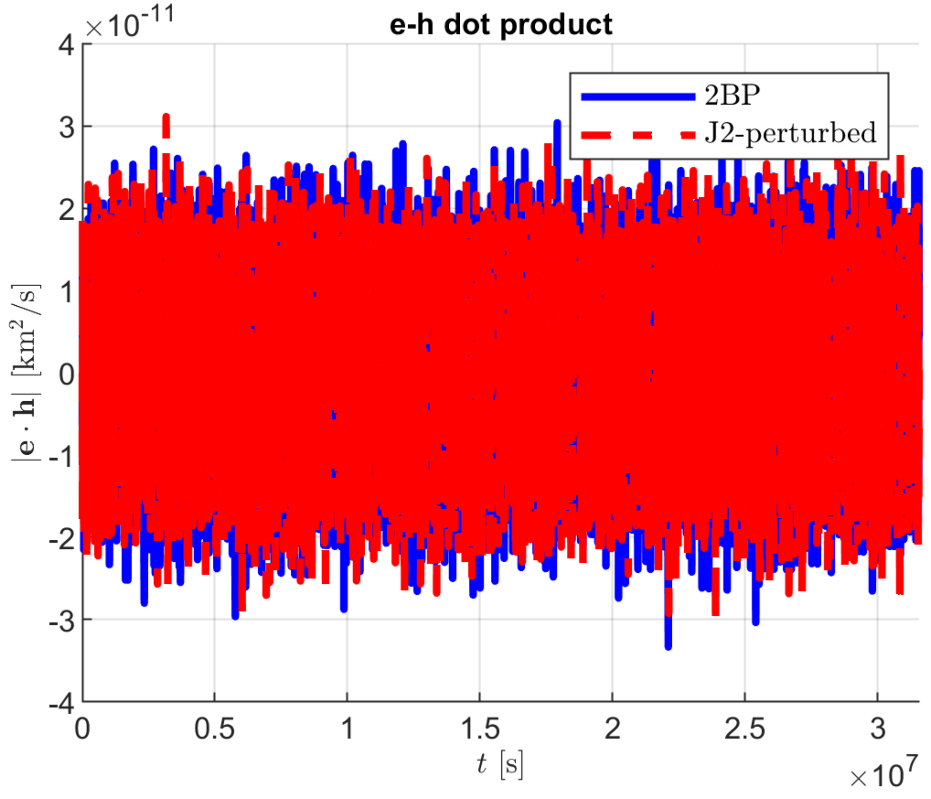
Parameters and initial condition

$$\mathbf{r}_0 = [\,26578.137, 0, 0\,]\ \text{km}$$

$$\mathbf{v}_0 = [\,0, 2.221, 3.173\,]\ \text{km/s}$$



Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$ – 5 periods



Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$ – 1 year

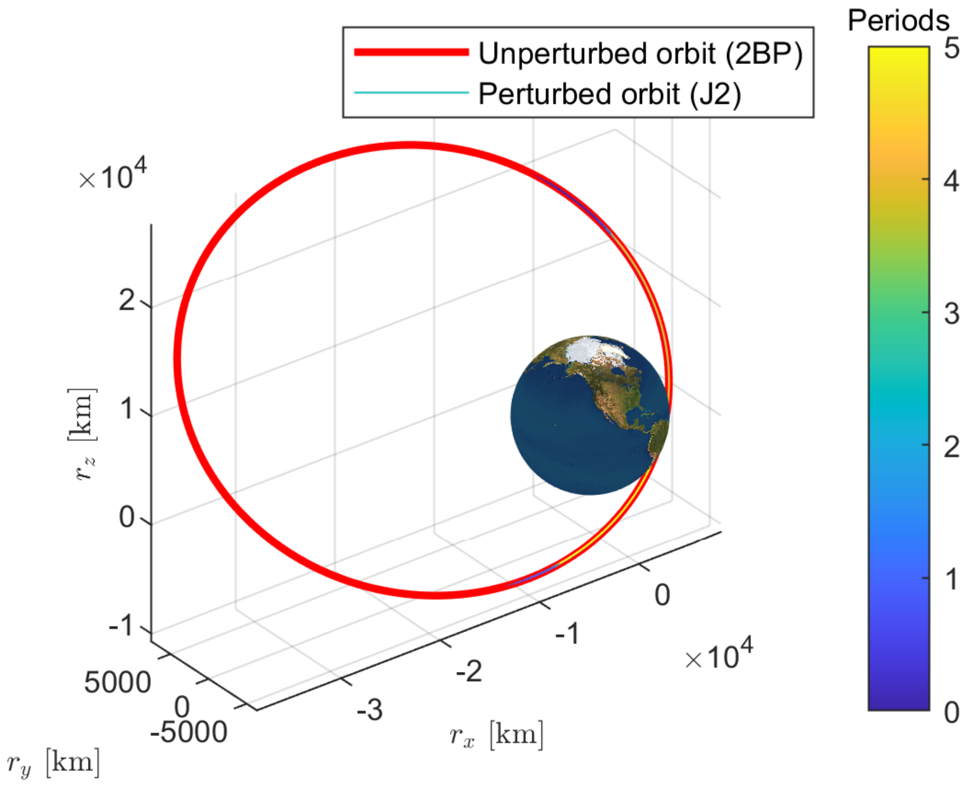
Exercise 2: Perturbed two-body problem

Sample solution: Highly eccentric and inclined orbit

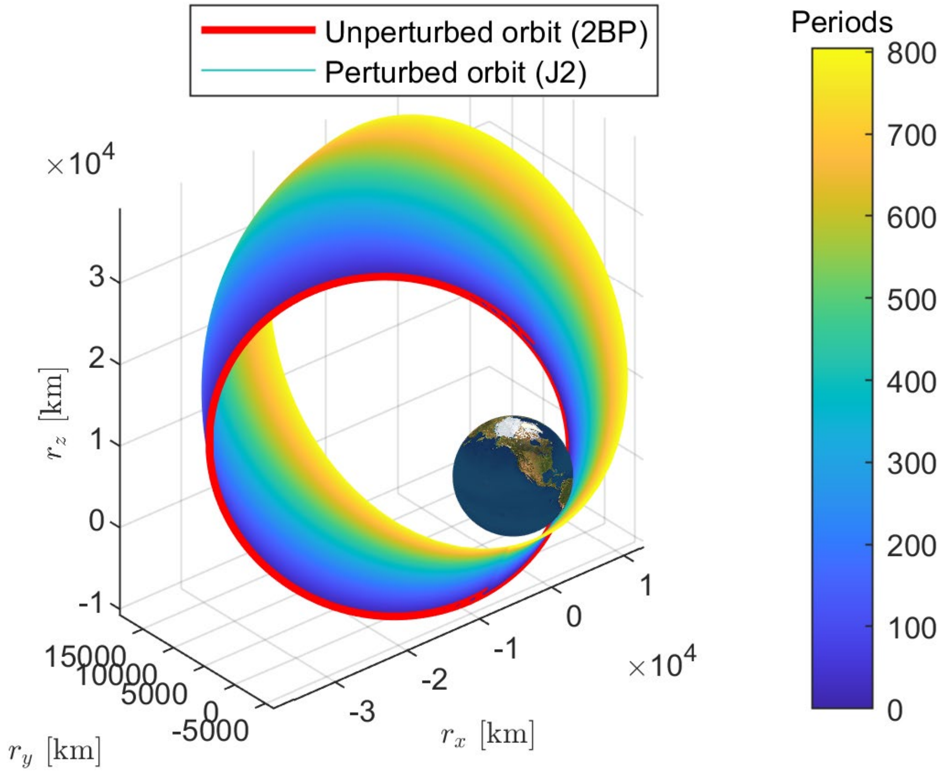
Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$

$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Orbit representation – 5 orbits



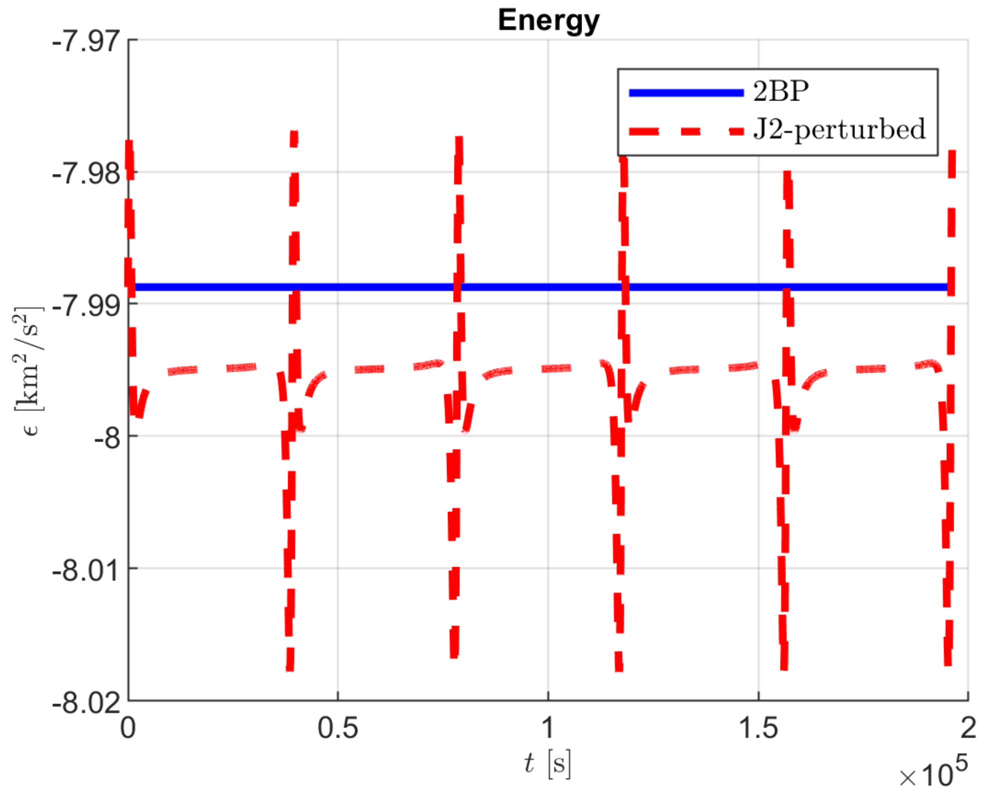
Orbit representation – 1 year

Exercise 2: Perturbed two-body problem

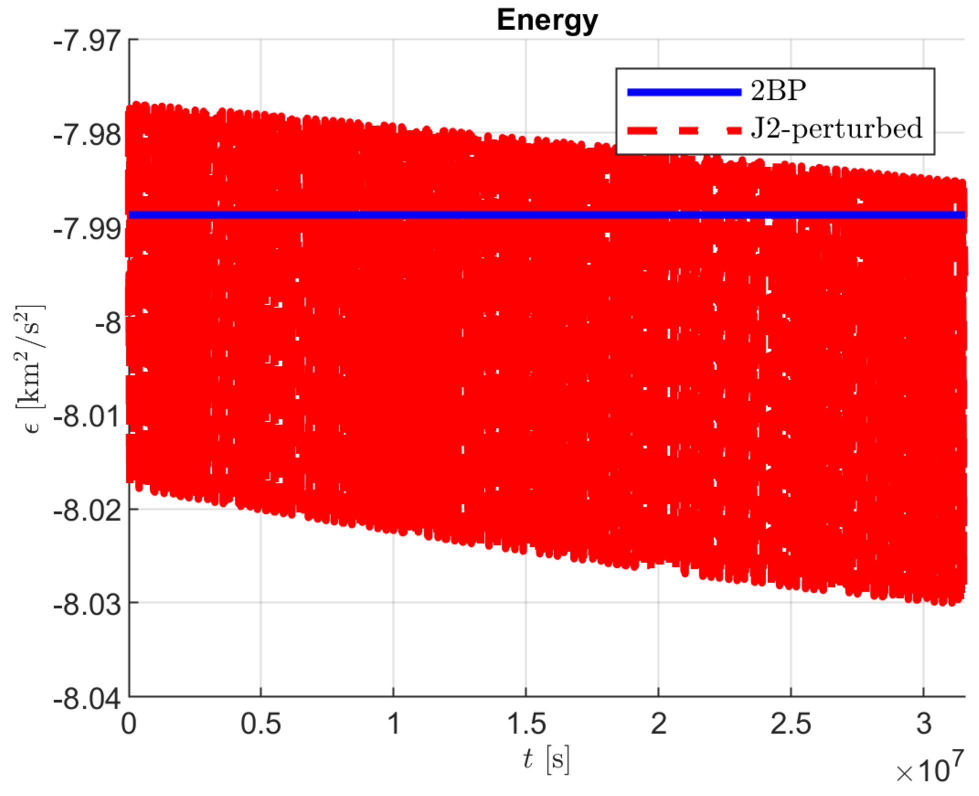
Sample solution: Highly eccentric and inclined orbit

Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622] \text{ km}$$
$$\mathbf{v}_0 = [4.752, 2.130, 7.950] \text{ km/s}$$



Specific energy – 5 orbits



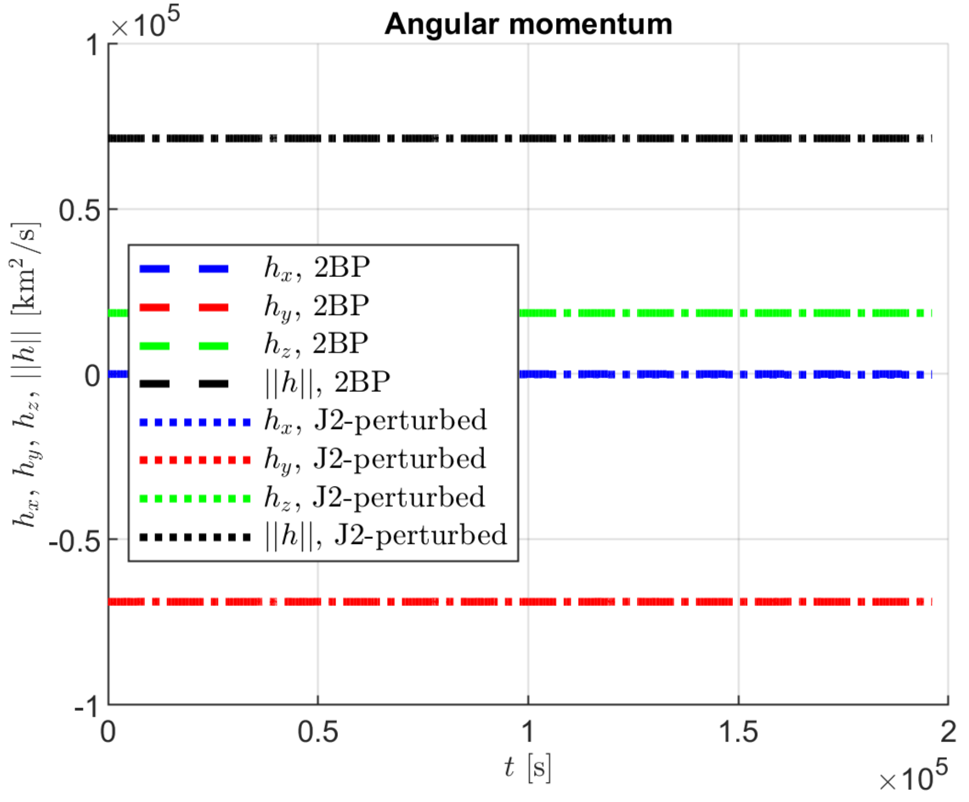
Specific energy – 1 year

Exercise 2: Perturbed two-body problem

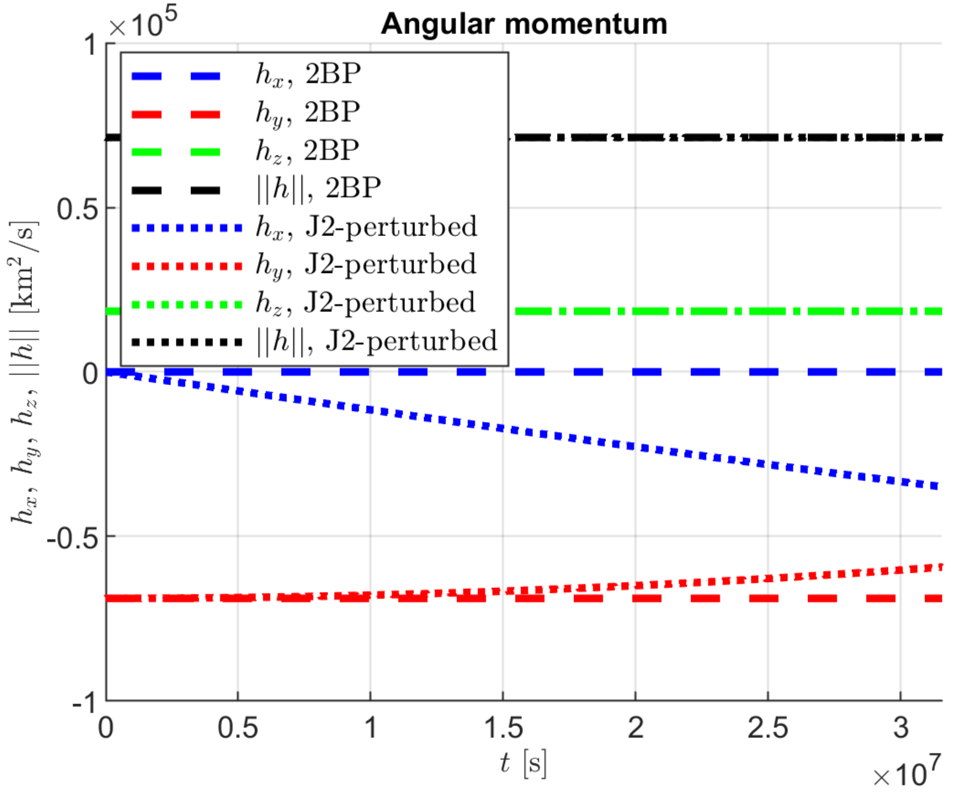
Sample solution: Highly eccentric and inclined orbit

Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$
$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Angular momentum – 5 orbits



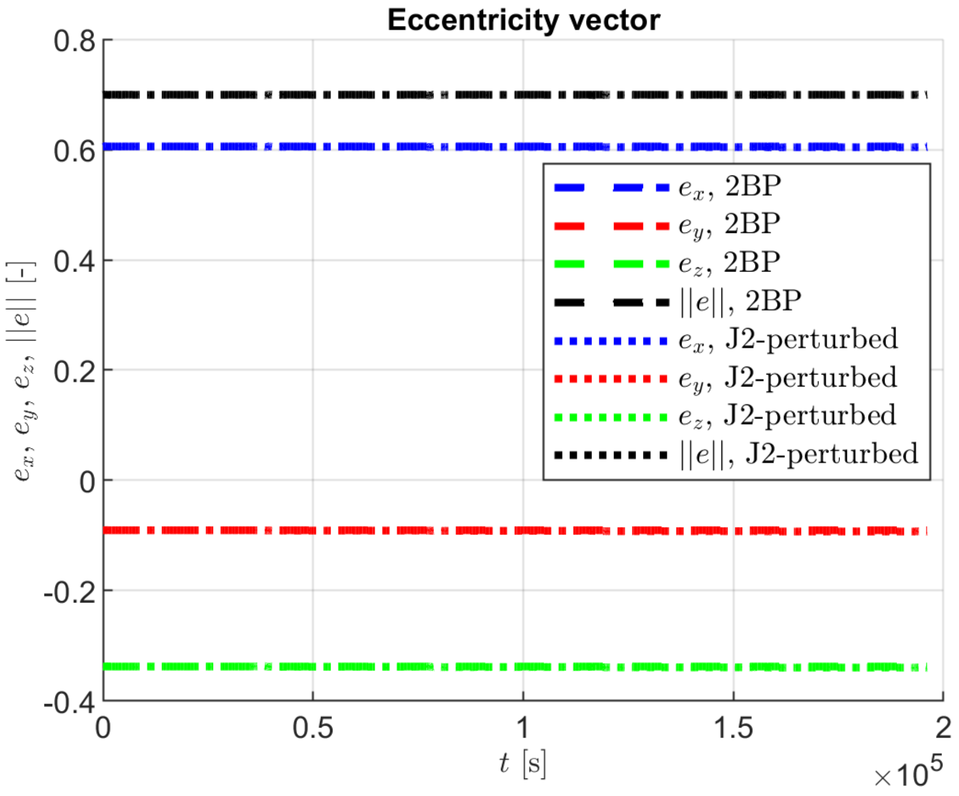
Angular momentum – 1 year

Exercise 2: Perturbed two-body problem

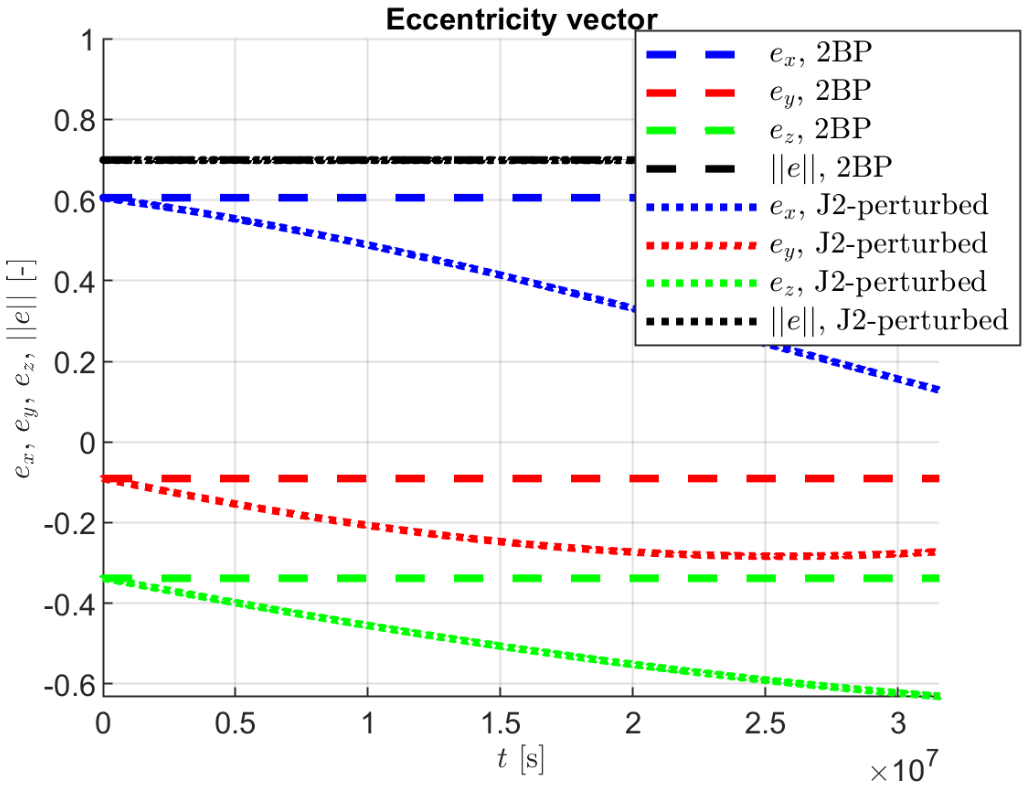
Sample solution: Highly eccentric and inclined orbit

Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$
$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Eccentricity vector – 5 orbits



Eccentricity vector – 1 year

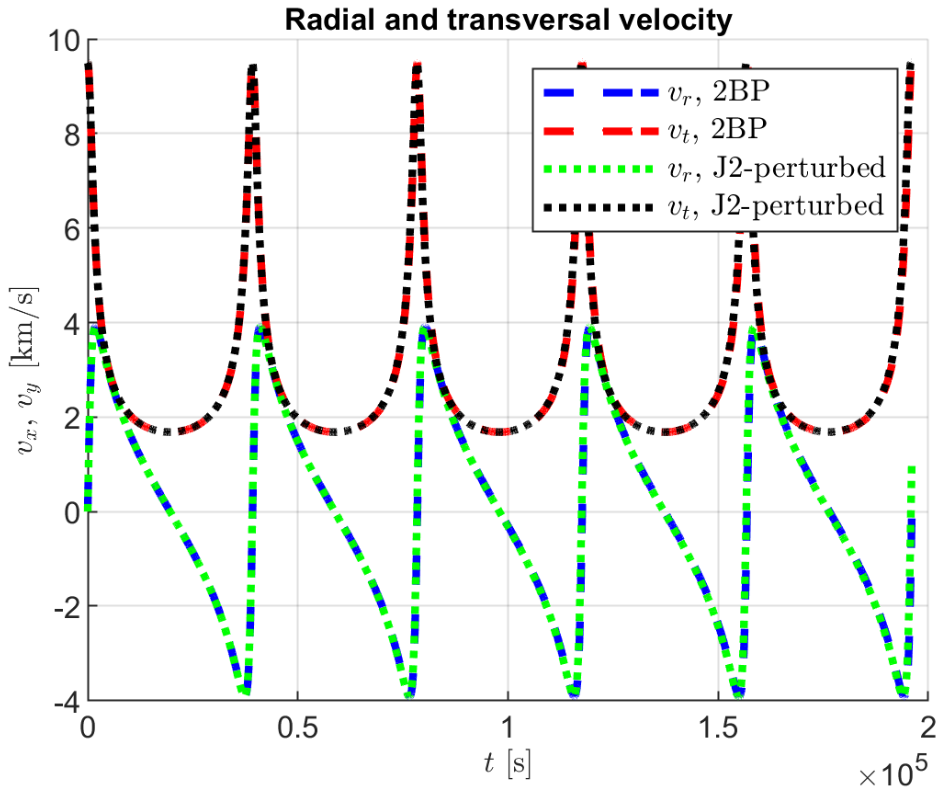
Exercise 2: Perturbed two-body problem

Sample solution: Highly eccentric and inclined orbit

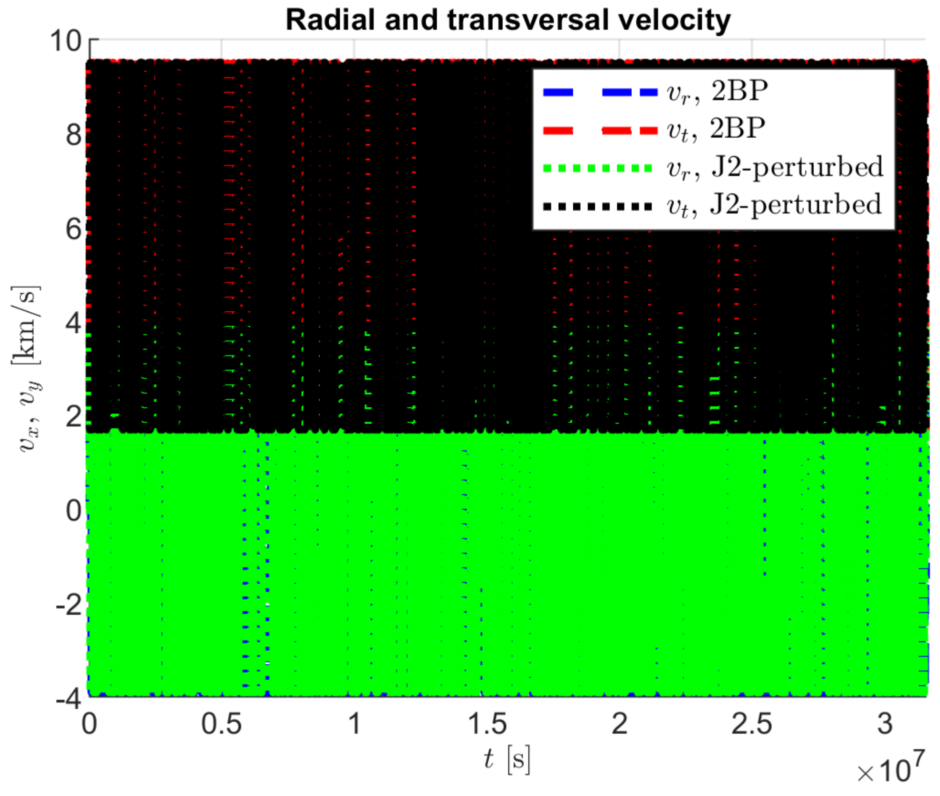
Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622]\ \text{km}$$

$$\mathbf{v}_0 = [4.752, 2.130, 7.950]\ \text{km/s}$$



Radial and transversal velocity – 5 orbits



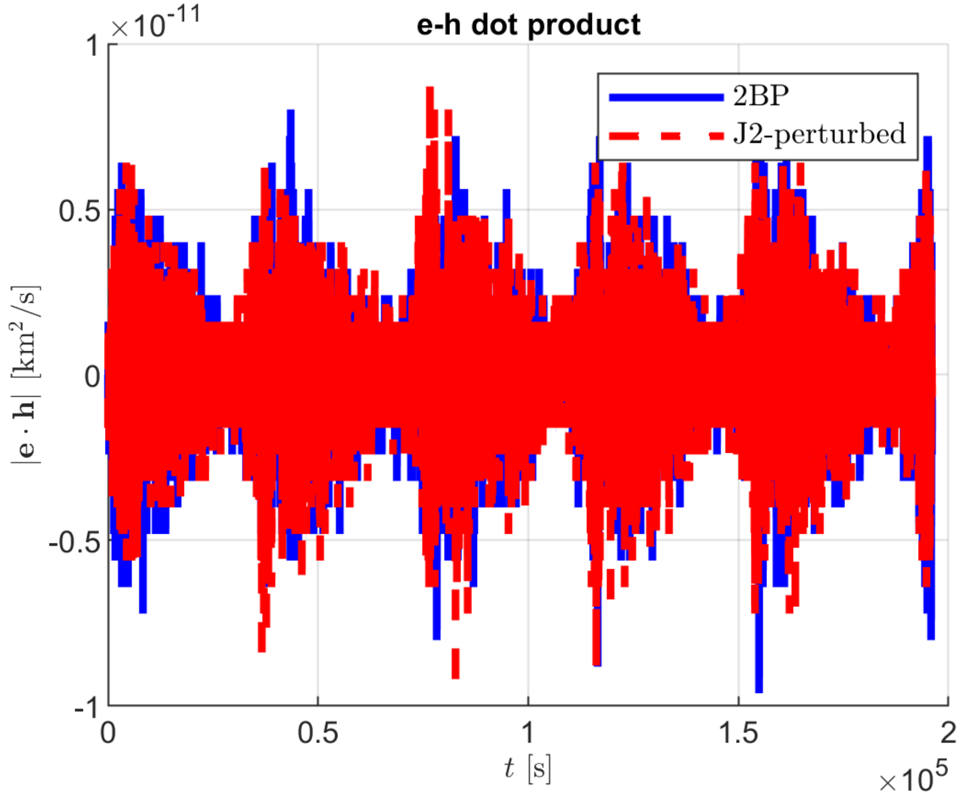
Radial and transversal velocity – 1 year

Exercise 2: Perturbed two-body problem

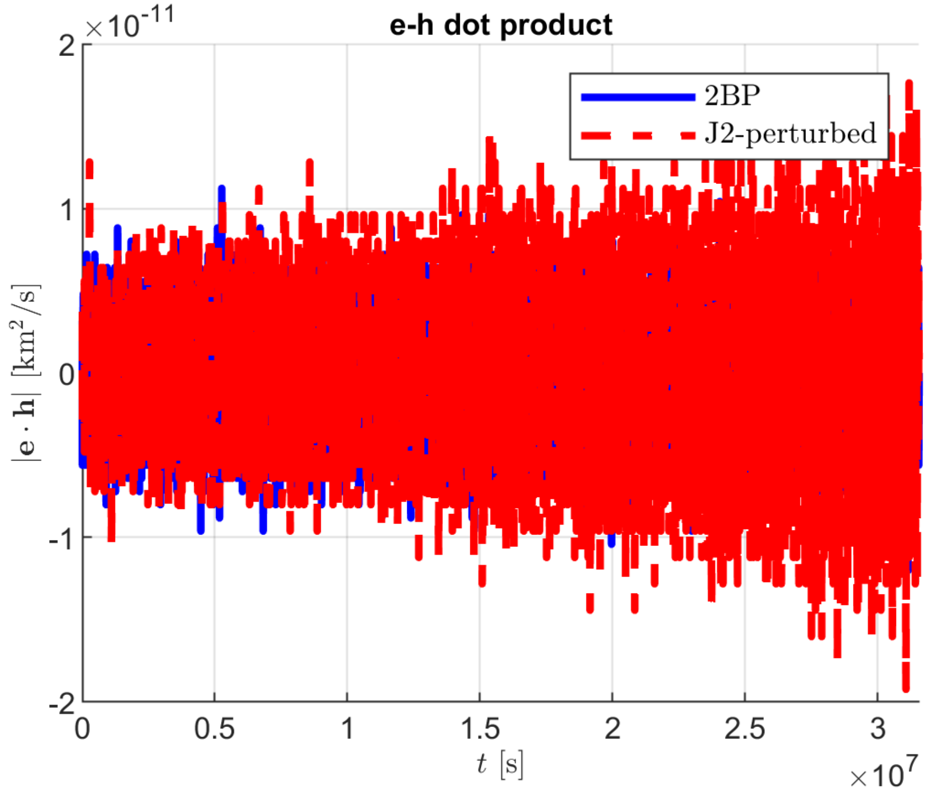
Sample solution: Highly eccentric and inclined orbit

Parameters and initial condition

$$\mathbf{r}_0 = [6495, -970, -3622] \text{ km}$$
$$\mathbf{v}_0 = [4.752, 2.130, 7.950] \text{ km/s}$$



Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$ – 5 orbits



Perpendicularity condition for $\mathbf{e}$ and $\mathbf{h}$ – 1 year

POLITECNICO
MILANO 1863

ROOT FINDING

Root-finding

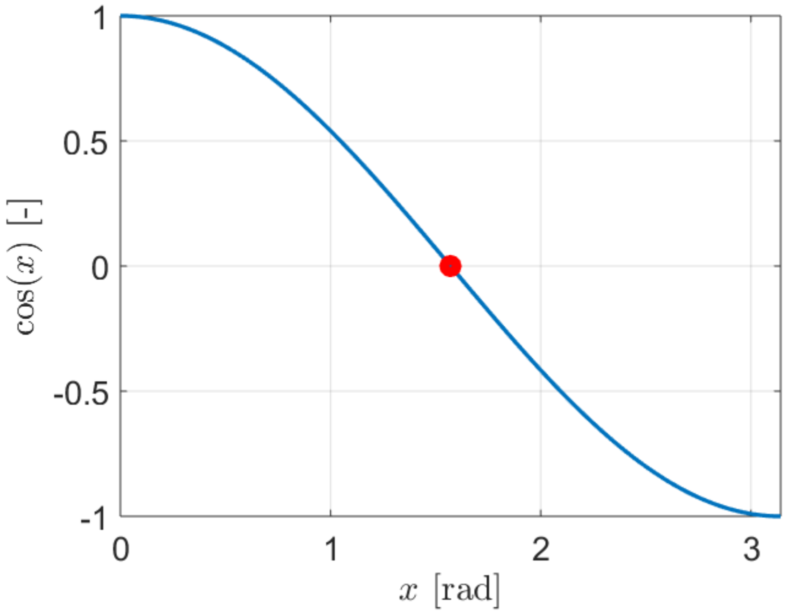
A basic problem in engineering

A root (or zero) of a function $\boldsymbol{F}(\boldsymbol{x})$ is a number $\boldsymbol{x_0}$ such that:

$$\boldsymbol{F}(\boldsymbol{x_0}) = \mathbf{0}$$

$\boldsymbol{F}$ and $\boldsymbol{x_0}$ can be scalars or vectors of the same dimension

- Root-finding is a very common problem in engineering
  - A classic Orbital Mechanics example is **solving Kepler's equation**
- There are many root-finding algorithms:
  - Bisection
  - Secant
  - Regula falsi
  - Newton's method
  - etc.



Root-finding

MATLAB’s root-finding algorithms

- **`fsolve`**
  - Can solve systems of non-linear equations
  - Includes several algorithms to choose from
  - Very robust and versatile
  - Part of the **`Optimization Toolbox`**

- **`fzero`**
  - Works only with scalar equations
  - Combines bisection, secant, and inverse quadratic interpolation methods
  - Can be faster than **`fsolve`**, especially if bounds for x_0 are provided

When the change of the function value in the last iteration is smaller in magnitude than `TolFun` or `FunctionTolerance` (default `1e-6`)

When the relative change of x in the last iteration is smaller in magnitude than `TolX` (default `2e-16`)

Termination Condition

Read the documentation pages on **`fsolve`** and **`fzero`** to learn how to use them!

Both require as inputs the function to be solved and an initial guess for the solution

- **Anonymous functions** can be helpful

Exercise 3: Kepler's equation

Kepler's equation: time law for the 2BP

- According to **first Kepler's law**, in the unperturbed 2BP the orbit is a conic. The shape and orientation of the orbit are constant, only the object's location inside the orbit changes in time.

- **Kepler's equation** provides a relation between the angular position E and time t, depending on the orbit's eccentricity e and mean motion $n = \sqrt{\mu/a^3}$. For elliptical orbits (i.e., with $e < 1$):

$$nt = E - e \sin E$$

  This is a **time law $\boldsymbol{E(t)}$ in implicit form** (we need a root-finding algorithm to solve it).

- The angular position repeats every period, so for times greater than one period we can discount the complete number of revolutions and solve Kepler's equation for the remaining time (improves numerical behaviour)
  1. Reduce t to a full number of orbits plus a remainder: $t = kT + \bar{t}, \quad k \in \mathbb{Z}$
  2. Solve Kepler's equation for $\bar{t}$: $\bar{E}(\bar{t})$
  3. Add the complete number of orbits: $E = k\, 2\pi + \bar{E}$

Exercise 3: Kepler's equation

Implement a solver for Kepler's equation

Exercise 3a: Implement a solver for Kepler's equation

- **Inputs:**
  - Time t
  - eccentricity e
  - semimajor axis a
  - gravitational parameter of the primary μ
  - reference initial time t_0 and eccentric anomaly E_0
- **Outputs**:
  - eccentric anomaly E
- Try and compare using both `fsolve` and `fzero`
- Be very careful with the ranges of the angles to prevent discontinuities

Exercise 3: Kepler's equation

Hints

- Read the documentation pages for `fsolve` and `fzero`
  - You may need to set the tolerance (error) for the solution
- You can use an anonymous function to pass the function to be solved to `fsolve` and `fzero`
- A good initial guess for the root is [2]: $E_{\text{guess}} = nt + \frac{e \sin nt}{1 - \sin(nt+e) + \sin nt}$
- You can add additional inputs as you see fit
  - It is possible to call a function without passing all the inputs. You can check the number of inputs passed using `nargin`. This way, you can program a function to have some inputs as 'optional'. Example:

```
% Seventh input is the tolerance 'tol', treated as optional
if nargin<7
    tol = 1e-6; % Default value used if input not given
end
```

[2] Battin, R., *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, 1999

Exercise 3: Kepler’s equation

Plot the evolution of eccentric anomaly with time

Exercise 3b: Plot $E(t)$ for different orbits

1. Create a function that computes the eccentric anomaly E for an orbit with fixed e, a, and μ, for an array of times with N-points covering k periods of the orbit
   - This new function can reuse the function you created in **Exercise 3a**
   - You can create arrays of uniformly distributed points in MATLAB using the **:** operator (colon) or the `linspace` function
   - The period of an orbit is $T = 2\pi/n$, where n is the mean motion
2. Compute $E(t)$ for the following values:
   - $a = 7000 \text{ km}$
   - μ_{Earth} from `astroConstants.m`
   - $E_0(t_0 = 0) = 0 \text{ rad}$
   - $k = 2, N \geq 100$
   - Six different eccentricities: $e = 0, 0.2, 0.4, 0.6, 0.8, 0.95$

Exercise 3: Kepler's equation

Plot the evolution of eccentric anomaly with time

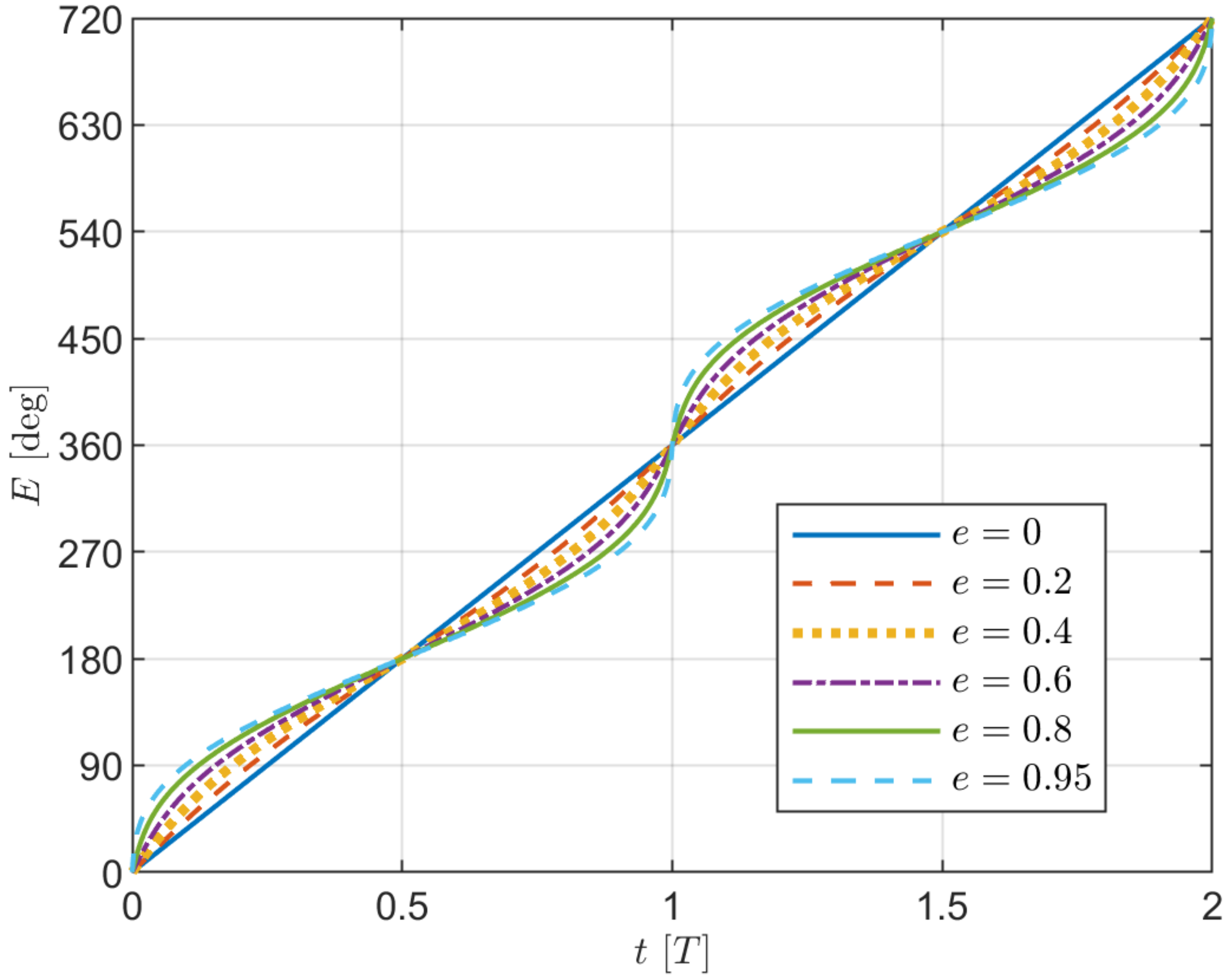
Exercise 3b: **Plot $E(t)$ for different orbits**

3. Plot $E(t; e)$ for all the cases in **2.** in a single 2D plot, using the `plot` command
   - Remember to set the axes and other properties of the plot
   - Hint: You can plot several lines at the same time with a single `plot` command, or you can use `hold on` to add them one by one

4. Plot $E(t; e)$ as a 3D surface using the `surf` command
   - Represent time in the x axis, eccentricity in the y axis, and eccentric anomaly in the z axis. Use the **documentation center** to check the correct way of passing these inputs to `surf`
   - You can compute $E(t; e)$ for additional values of e to get a smoother plot

Exercise 3: Kepler's equation

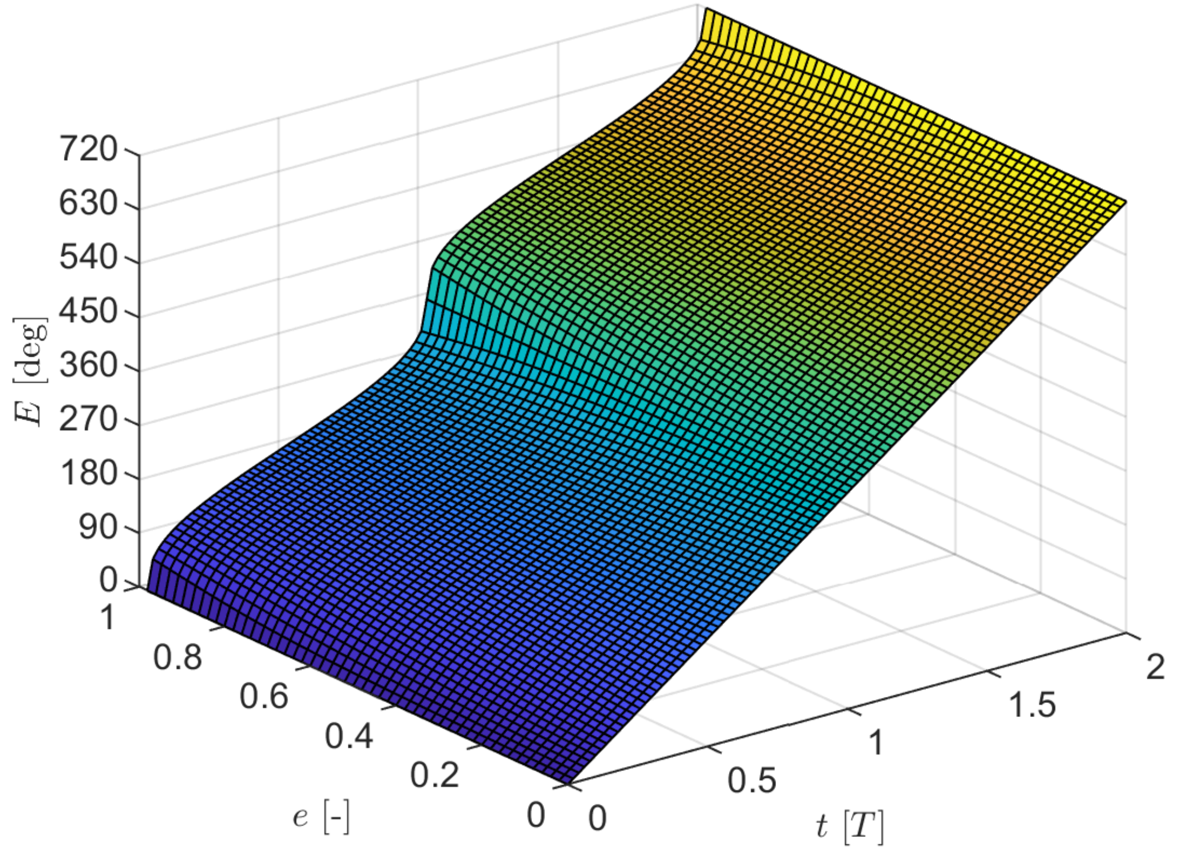
Solution: 2D plot

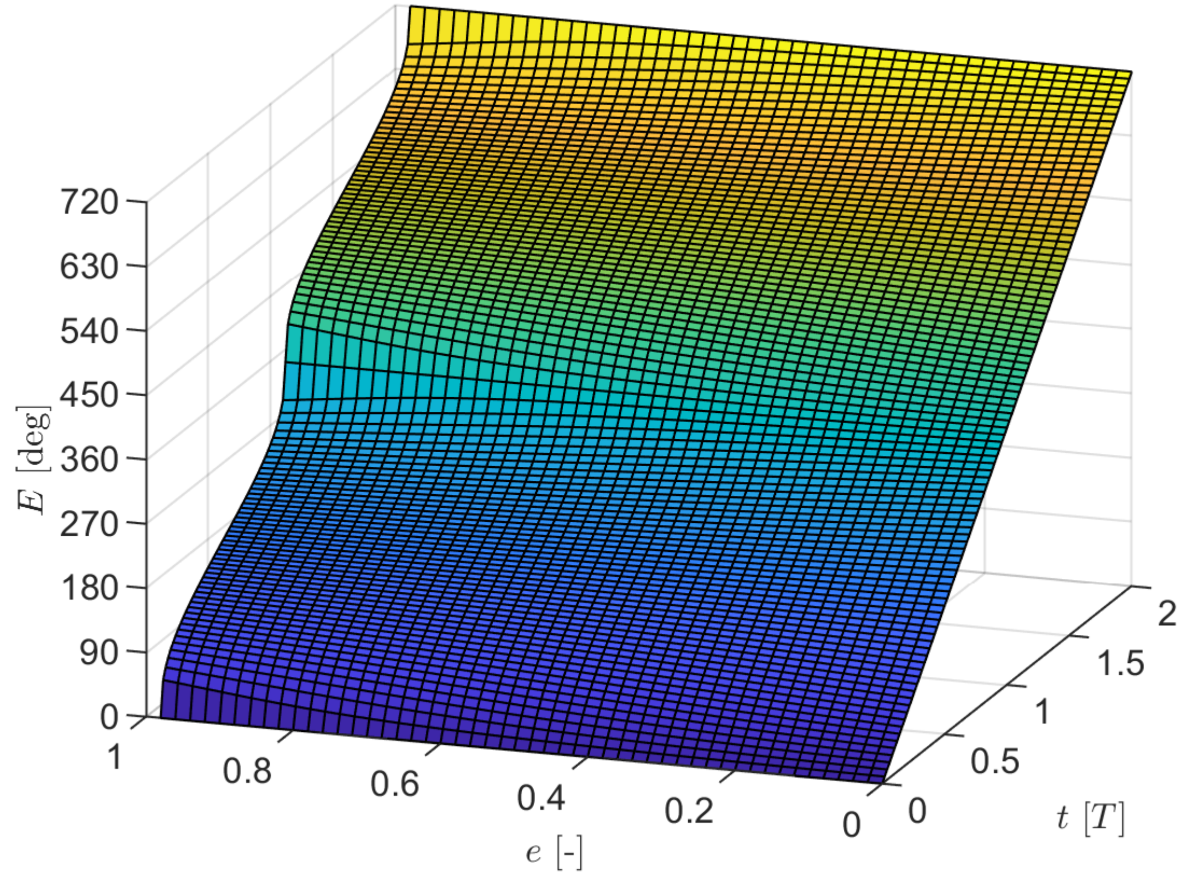


Evolution of eccentric anomaly with time for $a = 7000$ km, Earth as primary body, and different eccentricities

Exercise 3: Kepler's equation

Solution: surface plot





Evolution of eccentric anomaly with time and eccentricity for $a = 7000$ km and Earth as primary body

Additional exercise

Complete the form for a bonus in your final grade

After completing Exercises 1, 2 and 3, we ask you to **repeat the same analysis with the modified initial conditions** provided in this slide and **fill in the form** with the required information.

The form will close on **October 16th**.

This exercise is **not mandatory**, but correct answers will count as a **bonus in your final grade**.

https://forms.office.com/e/ME63vMDzsn

Additional exercise

Complete the form for a bonus in your final grade

Initial conditions, questions 1 and 2:

$$\mathbf{r}_0 = [242.9 \quad 5737.8 \quad 5809.1]\ \mathrm{km}$$
$$\mathbf{v}_0 = [-5.2090 \quad -3.2007 \quad 3.3786]\ \mathrm{km/s}$$

1. **Question 1:** evaluate the energy in the J2 perturbed two-body problem after 365 days [$\mathrm{km}^2/\mathrm{s}^2$]
2. **Question 2:** evaluate the eccentricity vector in the J2 perturbed two-body problem after 365 days [-]

Initial conditions, question 3:

$$\mathrm{a} = 7152\ \mathrm{km} \qquad \mathrm{e} = 0.95$$

All other parameters remain the same as Exercise 3

3. **Question 3**: compute the eccentric anomaly after 0.8 periods [deg]