July 7, 2017

Orbital Mechanics

Mission Analisys Final Report

POLITECNICO DI MILANO

DIPARTIMENTO DI
INGEGNERIA AEROSPAZIALE

Author: Alfonso Collogrosso

Author: Francescodario Cuzzocrea

Author: Benedetto Lui

In this report we will walk trough the steps that we have followed in order to cover in an optimal way the assigned transfers. The first chapter will be about a transfer from the Earth to the Florance 35 asteroid, while the second chapter will look at an interplanetary transfer with a flyby on Saturn between Mars and Neptune. The third chapter will cover a preliminary analysis about perturbations on a given initail orbit

Website:
https://github.com/fcuzzocrea/OrbitalMechanics2016

Contents

Chapter 1

Orbital Maneuvers

1.1 Introduction

The aim of this chapter is to explain the design of an orbital transfer between the Earth and the asteroid Florence 35, an Apollo class asteroid. The Apollo asteroids are a group of NEO asteroids that have an orbit characterized by a a semi major axis greater than the one of the Earth; the perihelion distance instead, is lesser than Earth's aphelion distance. As main consequence of this fact, they cross the Earth's orbit around the Sun.

The procedure applied in order to design the transfer is divided in three main steps:

1. Set all known parameters and computation of Earth's and Florence's orbits;
2. Solve the Lambert's problem for every possible combination of TOFs at intervals of one week;
3. Evaluation of the best Lambert's arc.

A suboptimal solution is then chosen in order to respect the assigned launcher's C3 constraint. A qualitative comparison with a classical Hohmann transfer is then carried out in order to evaluate the differences in terms of ΔV between the two solutions.

The assigned windows for departure from the Earth and arrival to Florence are reported in the following table :

Target	T1	T2	V_∞
Florence (35)	2024/31/1 - 2027/1/1	2024/11/1 - 2029/6/1	5.8

Table 1.1: Given data

In figure 1.1 the path of the Florence's orbit and the Earth's orbit in an heliocentric around the sun is shown :

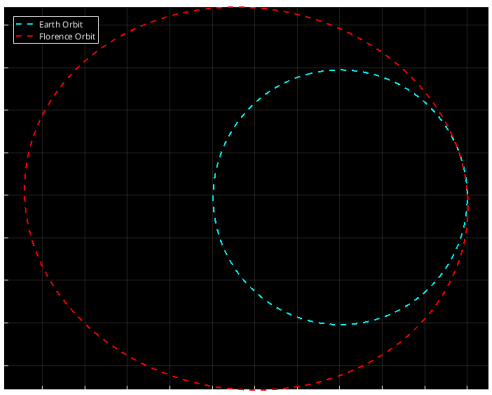


Figure 1.1: Earth's and Florence 35 orbits

The Lambert's Problem

For the Lambert's problem, two position vectors and the time of flight between each of the two are known, but the orbit between the endpoints is unknown.

The MATLAB function that solves the Lambert's problem, given initial and final position vector, and gives as outputs the orbital parameters of the transfer ellipse is called at every step of the main routine. Once the initial velocity and the final

velocity of the Lambert's arc are evaluated by the function, the total cost of the transfer ellipse in terms of ΔV can be easily evaluated by the following calculation:

$$\Delta V_a = V_{trans_A} - V_{sat}$$

$$\Delta V_b = V_{sat} - V_{trans_B}$$

$$\Delta V_{TOT} = |\Delta V_a| + |\Delta V_b|$$

with respect to figure 1.2:

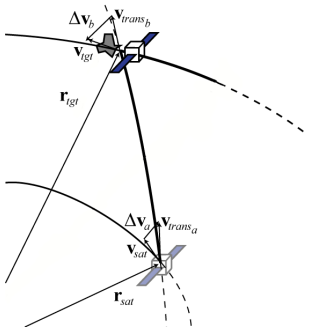


Figure 1.2: Lambert's Arc

1.2 Design Strategy

In order to obtain the best orbital transfer we have developed a MATLAB script that, once all the possible combinations of TOFs are evaluated, launches a main routine in which the Lambert's arcs are then evaluated for each possible combinations of departure time and arrival time, at intervals of one week.

TOFs Calculation

Since the computation should be done for every possible combination of departure and arrival dates, a function has been developed to basically build a matrix with all the TOFs in the following way:

Algorithm 1.1 TOFs Matrix Computation

```
function [TOF] = tof_calculator (t_dep_vect,t_arr_vect)

% Preallocation
l1 = length(t_dep_vect);
l2 = length(t_arr_vect);
TOF = zeros(l1,l2);

% For cycle to obtain the TOF matrix
for i = 1 : l1
        TOF(i,:) = t_arr_vect - t_dep_vect(i);
end
end
```

Obviously, all the negative TOFs are automatically set to NaN in the main script so they aren't processed in the main routine. The TOF matrix is then passed to the main routine in order to solve the Lambert's problem.

The Main Routine

The main routine has been implemented as it's shown in Algorithm 1.2:

Algorithm 1.2 ΔV Matrix Computation

```
for i = 1 : length(t_dep)
        r_e = r_dep_vect(i,:);
        v_e = v_dep_vect(i,:);
                for j = 1 : length(t_arr)
                        tof = TOF_matrix(i,j)*86400;
                        if tof > 0
                                r_f = r_arr_vect(j,:);
                                v_f=v_arr_vect(j,:);
                                [~,~,~,~,VI,VF,~,~]=lambertMR(r_e,r_f,tof,ksun);
                                dv1 = norm(VI - v_e);
                                dv2 = norm(v_f - VF);
                                Dv_matrix(i,j)=abs(dv1)+abs(dv2);
                                v_inf_matrix(i,j) = dv1;
                        else
                                Dv_matrix(i,j) = nan;
                                v_inf_matrix(i,j) = nan;
                        end
                end
end
```

As we can see, the for cycle passes through each day and computes the Lambert's arcs for all the combinations of launch date and arrival date. Then the total ΔV for each one of the arcs is computed and stored in a 2D matrix.

The Pork Chop Plot

A pork chop plot is a data visualization tool that can be used in interplanetary mission design, because it displays contours of various quantities as a function of departure and arrival date. For the Earth-Florence transfer we get the pork chop plot shown in figure 1.3,

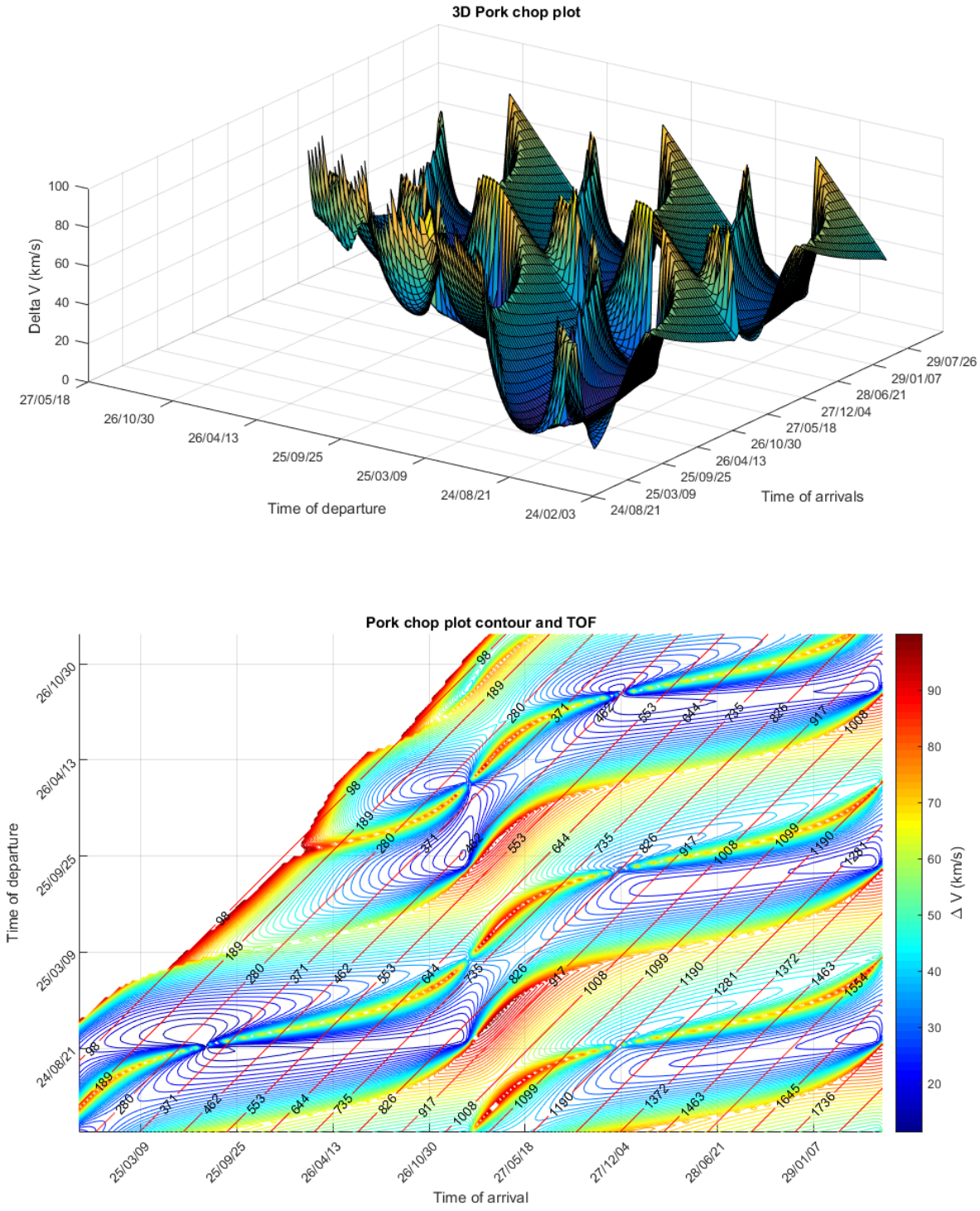


Figure 1.3: Pork Chop Plot

in which the x-axis is the Earth departure date, the y-axis is the Florence arrival date. The contour lines show the sum of Earth departure ΔV and Florence arrival ΔV, which are related to the C3, the "characteristic energy" of the departure or arrival. So, each point on the curve represents a transfer (computed using the Lambert solver shown in the previous section). A given contour, called a pork chop curve, represents constant ΔV_{TOT}, and the center of the pork chop the optimal minimum ΔV_{TOT}, and thus the optimal transfer trajectory. On top of the Pork Chop Plot we have also superimposed the TOFs constant lines that are visible in red. As we can see from the pork chop plot, there are different zones of minimum which can be selected as possible departure window and arrival window.

Best Transfer Arc

The best ΔV can be extracted from theΔV matrix by simply taking is minimum, then, using the MATLAB function **find**, we can retrieve its position in terms of rows and columns in the matrix itself so we can retrieve departure position and arrival position from its respective vectors:

Algorithm 1.3 Minimum ΔV

```
Dv_min = min(min(Dv_matrix));

[ROW,COLUMN] = find(Dv_matrix == Dv_min);
Dv_min_TOF = (TOF_matrix(ROW,COLUMN)*86400);
r1_arc = r_dep_vect(ROW,:);
r2_arc = r_arr_vect(COLUMN,:);
```

As result we can see that the minimum ΔV is about 11.277354 km/s, while the maximum ΔV is about 99.997552 km/s.
The transfer arc that corresponds to the minimum ΔV can now be integrated and its path is shown in figure 1.4 in yellow

Algorithm 1.4 Best Transfer Arc Integration

```
[~,~,~,~,v1_arc,v2_arc,~,~] = lambertMR(r1_arc,r2_arc,Dv_min_TOF,ksun);
[rx_arc, ry_arc, rz_arc, vx_arc, vy_arc, vz_arc] = intARC_lamb(r1_arc,...
...v1_arc,ksun,Dv_min_TOF,86400);
```

We can also compute a suboptimal transfer arc (that is the one in green in figure 1.4) by taking the V_∞ =5.789149 km/s that is the value in the ΔV matrix that is nearest to the given V_∞. As a result we get a ΔV of 12.0893 km/s, that is slightly larger than the one found for the best transfer arc. As result also the path is more or less equal, as it can be seen from figure 1.5.

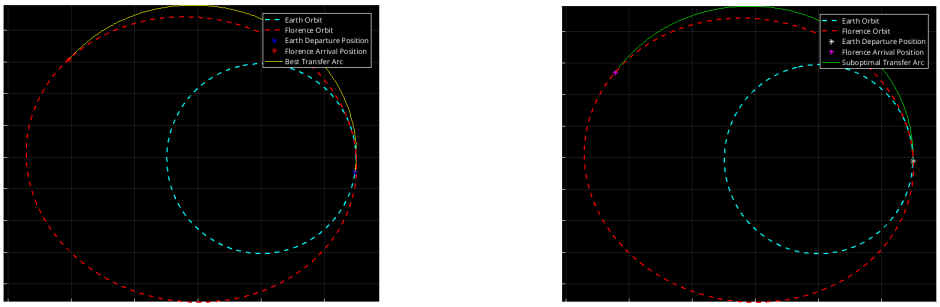


Figure 1.4: Transfer Arcs

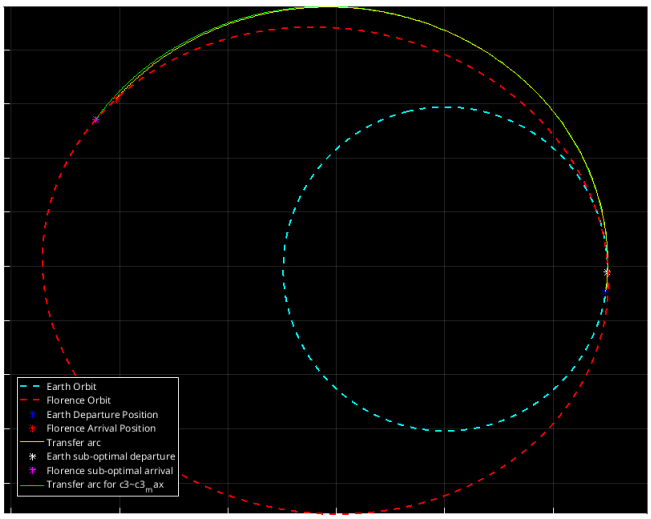


Figure 1.5: Best Transfer Arc and Suboptimal Transfer Arc

Hohmann Transfer Comparison

To do the Hohmann transfer comparison, the MATLAB functions `hohmann.m`, `inclinationchange.m` and `anoperichange.m` have been developed, and are then called in the main script in order to compute the cost of an hypothetical Hohmann transfer from the Earth to Florence asteroids.

Since the two orbits are on different planes, a change of inclination i is needed, and then a change of the anomaly of the pericenter ω is done in order to put the spacecraft in the Florence orbit, since the two orbits are not coaxial. In order to reach the maximum efficiency in terms ofΔV, the change of inclination is done after the change of semi major axis.

However, as expected, the overall cost of an Hohmann transfer is greater than the Lambert's arc strategy due to the change of plane, in fact the results are:

	ΔV [km/s]
Change of a	5.9186
Change of i	11.3541
Change of ω	0.7596
Total Cost	18.032333

Table 1.2: Hohmann Transfer

Chapter 2

Interplanetary Flyby

2.1 Introduction

The aim of this chapter is to explain the design of an interplanetary trajectory from Mars to Neptune with a flyby on Saturn.

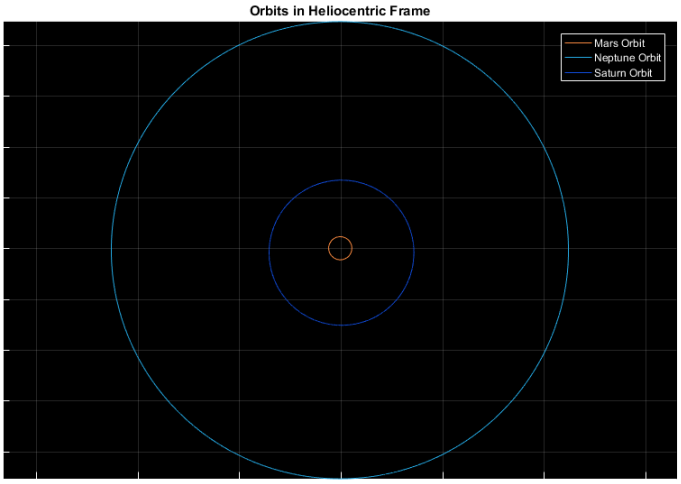


Figure 2.1: Given Orbits

The procedure applied in order to design the transfer is divided in four main steps:

1. Set all known parameters and computation of orbits for all of the three planets;
2. Solve the Lambert's problem for the selected combination of Mars departure time, Saturn flyby time and Neptune arrival time;
3. Evaluation of the powered gravity assist;
4. Evaluation of the more energy efficient Lambert's arcs.

The given time window goes from January 1, 2016 to December 31, 2055.

The method of the patched conics

Interplanetary mission no longer conform to the idealized two-body problem. Just the fact that some vehicle is attempting to leave one planet to traverse the solar system, which revolves around the Sun, in order to reach another planet introduces at least three bodies. To make matters worse, interplanetary travel is also subject to the forces of other planets and objects which may be near to or in between the initial and final locations.

In order to simplify interplanetary missions design, a method known as patched (or linked) conics method has been developed. It basically breaks the multy-body system into three different sections of two body problems. We define the sphere of influence (from now on SOI) of the planet as an attempt to mark the location where the Sun's gravitational effects become dominant with respect to the planet's one. We assume that when the spacecraft is outside the SOI of a planet it follows an unperturbed keplerian orbit around the sun.

So the method is split into three segments :

- Initial planet's SOI where the spacecraft leaves planet's on hyperbolic escape trajectory;
- Elliptical trajectory about the sun from initial planet towards the final planet;
- Entering into the final planet's SOI on an hyperbolic capture trajectory and maneuvering in order to enter into an elliptical parking orbit around the final planet or descend to the planet surface, or flyby to another destination.

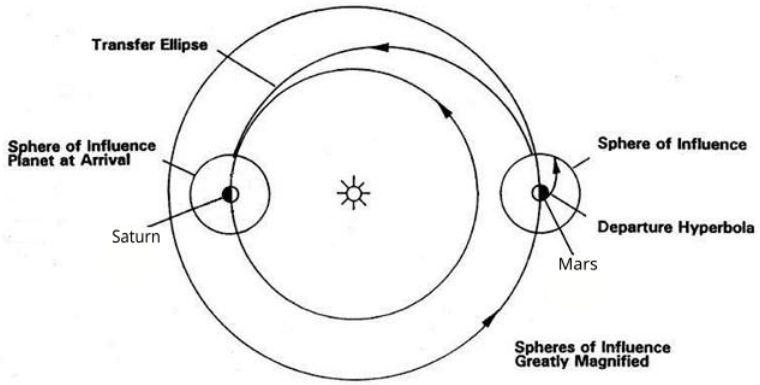


Figure 2.2: Method of Patched Conics

The expression of the SOI of a planet orbiting the sun is given by:

$$R_{SOI} = R_{planet} \left(\frac{m_{planet}}{m_{sun}} \right)^{2/5}$$

where R_{planet} is the radius that goes from the Sun to the planet.

Although this method gives a good approximation of trajectories for interplanetary spacecraft missions, there are missions for which this approximation does not provide sufficiently accurate results.

2.2 Design Strategy

Iterative routine

To obtain the most energy efficient flyby, as a first design strategy an iterative routine has been developed. First all the possible TOFs are evaluated, then the main routine starts cycling around every selected day: all the Lambert's arcs for each possible combinations of departure from Mars and arrival to Saturn and departure from Saturn and arrival to Neptune are evaluated at interval of 200 days. The 200 days limit it is imposed due to computing costraint. Then the flyby is computed for the best energy efficient combination of arrival/departure to/from Saturn.

TOFs Calculation

The TOFs matrix has been evaluated in the same way as in section 1.2. The resultant TOFs matrix is triangular because half of evaluated TOFs can't be accepted as the departure time window and the arrival time window are the same, so half of the matrix contains negative time intervals. Also the situation in which the departure from Saturn to Neptune happens before the departure from Mars to Saturn should be avoided.

The Main Cycle

The main routine has been developed by slightly modifying the algorithm 1.2 showed in section 1.2:

Algorithm 2.1 ΔV Matrix Computation for Flyby

```
% Computation of the 3D-Tensor of deltav with 3 nested for cycles

for i = 1:length(t_dep)
        r_mars = r_dep_vect_mars(i,:);
        v_mars = v_dep_vect_mars(i,:);
        for j = 1:length(t_dep)
                tof_1 = TOF_matrix(i,j)*86400;
                if tof_1 > 0
                        r_saturn = r_dep_vect_saturn(j,:);
                        v_saturn = v_dep_vect_saturn(j,:);
                        [~,~,~,~,VI_mars,VF_saturn,~,~] = lambertMR(r_mars,r_saturn,tof_1,ksun);
                        dv1_mars = norm(VI_mars - v_mars);
                        dv2_saturn = norm(v_saturn - VF_saturn);
                        Dv_matrix_1(i,j) = abs(dv1_mars) + abs(dv2_saturn);
                        v_inf_matrix_1(i,j) = dv1_mars;
                        for k = 1:length(t_dep)
                                tof_2 = TOF_matrix(j,k)*86400;
                        if tof_2 > 0
                                        r_neptune = r_dep_vect_neptune(k,:);
                                        v_neptune = v_dep_vect_neptune(k,:);
                                        [~,~,~,~,VI_saturn,VF_neptune,~,~] = ...
                                        ... = lambertMR(r_saturn,r_neptune,tof_2,ksun);
                                        dv1_saturn = norm(VI_saturn - v_saturn);
                                        dv2_neptune = norm(v_neptune - VF_neptune);
                                        Dv_matrix_2(j,k) = abs(dv1_saturn) + abs(dv2_neptune);
                                        v_inf_matrix_2(j,k) = dv1_saturn;
                                        dv_ga = powerflyby(dv2_saturn,dv1_saturn, ...
                                                astroConstants(ibody_saturn+10));
                                        DV_Tensor(i,j,k) = Dv_matrix_1(i,j)+dv_ga+Dv_matrix_2(j,k);

                                else
                                        Dv_matrix_2(j,k) = nan;
                                        v_inf_matrix_2(j,k) = nan;
                                        DV_Tensor(i,j,k) = nan;
                                end
                        end
                else
                        Dv_matrix_1(i,j) = nan;
                        v_inf_matrix_1(i,j) = nan;
                        DV_Tensor(i,j,:) = nan;
                end
        end
end
```

as it can be seen from algorithm 2.1 there are three nested for cycle that passes trough each selected departure day. The second cycle calculates the Lambert's arc that goes from Mars to Saturn, while the third cycle calculates the arc that goes from Saturn to Neptune, and the ΔV needed to "power" the flyby, throug the function `powerflyby`, described afterwards. The total ΔVs that is computed by summation of the ΔVs required for both arcs are then stored on a 3D tensor. If during the cycle any TOF is negative will be automatically skipped. With respect to the previous ΔV matrix (the one of section 1.2), this time the matrix will be a 3D matrix because of the fact that all the possible combinations are now taken for two Lambert's arcs, not for one.

The pork chop plots for both arcs reflect the facts that half of the window can't be used to avoid a departure before of an arrival :

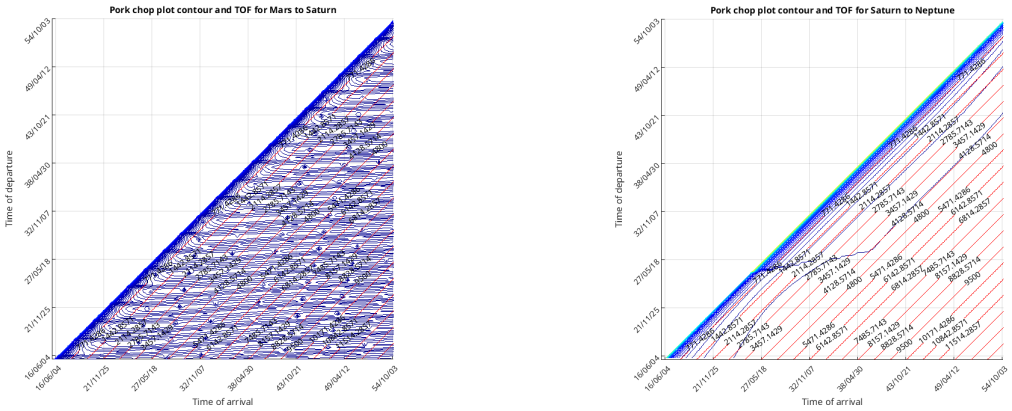


Figure 2.3: Pork Chop Plots of the Two Paths

Transfer Arcs

The more energy efficient ΔVs for both arcs can be extracted from the ΔV matrix by simply taking is minimum. In order to retrieve their position in terms of rows, columns and "depth" inside the ΔV matrix the MATLAB command `ind2sub` is used before the **`find`** command, due to the fact that **`find`** does not work on 3D matrices :

Algorithm 2.2 Best Transfer Arcs

```
% Find the minimum DV
DV_MIN = min(min(min(DV_Tensor)));
DV_MAX = max(max(max(DV_Tensor)));
[row,column,depth] = ind2sub(size(DV_Tensor),find(DV_Tensor == DV_MIN));

% Find best arcs
r1_arc = r_dep_vect_mars(row,:);
r2_arc = r_dep_vect_saturn(column,:);
r3_arc = r_dep_vect_neptune(depth,:);

% Find correspondent TOFs
Dv_min_TOF_1 = (TOF_matrix(row,column)*86400);
Dv_min_TOF_2 = (TOF_matrix(column,depth)*86400);
```

The minum ΔV found with this method is $11.2869\,\mathrm{km/s}$.

Fmincon optimizator

In order to obtain a ΔV lower than one find by the iterative routine, a possible solution with the MATLAB `fmincon` optimizer has been investigated. In order to use such optimizer, a fitness function, that basically implements the same steps implemented in the iterative routine, has been developed. The fitness function is evaluated at each step by the optimizer in order to find the best days that are the parameter of the fitness function itself. The fitness function has been implemented has follows :

Algorithm 2.3 Fitness Function

```
function [Dv, r_mars, r_saturn, r_neptune, v_saturn, dv_ga] = dv_optimizator(x)

% From ephemeris compute position and velocity for the selected day
[kep_dep_vect_mars,ksun] = uplanet(x(1),ibody_mars);
[r_mars,v_mars] = kep2car(kep_dep_vect_mars,ksun);
[kep_dep_vect_saturn,~] = uplanet(x(2),ibody_saturn);
[r_saturn,v_saturn] = kep2car(kep_dep_vect_saturn,ksun);
[kep_dep_vect_neptune,~] = uplanet(x(3),ibody_neptune);
[r_neptune,v_neptune] = kep2car(kep_dep_vect_neptune,ksun);

% TOF to go from Mars to Saturn, in days
tof_1 = (x(2)-x(1))*86400;

% Compute the Lambert arc for tof_1
[~,~,~,~,VI_mars,VF_saturn,~,~] = lambertMR(r_mars,r_saturn,tof_1,ksun);
dv1_mars = norm(VI_mars - v_mars');
dv2_saturn = norm(v_saturn' - VF_saturn);

% TOF to go from Saturn to Neptune, in days
tof_2 = (x(3)-x(2))*86400;

% Compute the Lambert arc for tof_2
[~,~,~,~,VI_saturn,VF_neptune,~,~] = lambertMR(r_saturn,r_neptune,tof_2,ksun);
dv1_saturn = norm(VI_saturn - v_saturn');
dv2_neptune = norm(v_neptune' - VF_neptune);

% "Powered" DV
dv_ga = powerflyby(dv2_saturn,dv1_saturn,astroConstants(ibody_saturn+10));

% Compute the total DV
Dv = dv1_mars + dv_ga + dv2_neptune;
end
```

The optimizer is called by a function called in the main script. As upper bound is given the last day of the window, as lower bound the first day of the window. As costraint, as usual, we have that the arrival must be after the departure.

The minum ΔV found with this method is $12.8127\,\mathrm{km/s}$.

Genetic algorithm

In order to find the minimum possibleΔV the MATLAB genetic algorithm implementation `ga` has been investigated. The fitness funcion, the upper bounds and the costraints are the same of the `fmincon` optimizer. Due to te randomatic behaviour of the genetic algorithm, we have developed an iterative cycle that calls `ga` with a maximum population of 100 elements several times, in order to search for the minimumΔV present in the resulting set of ΔVs.

After this optimization process we have found a minimum ΔV of $10.61\,\mathrm{km/s}$.

2.3 Hyperbola parameters

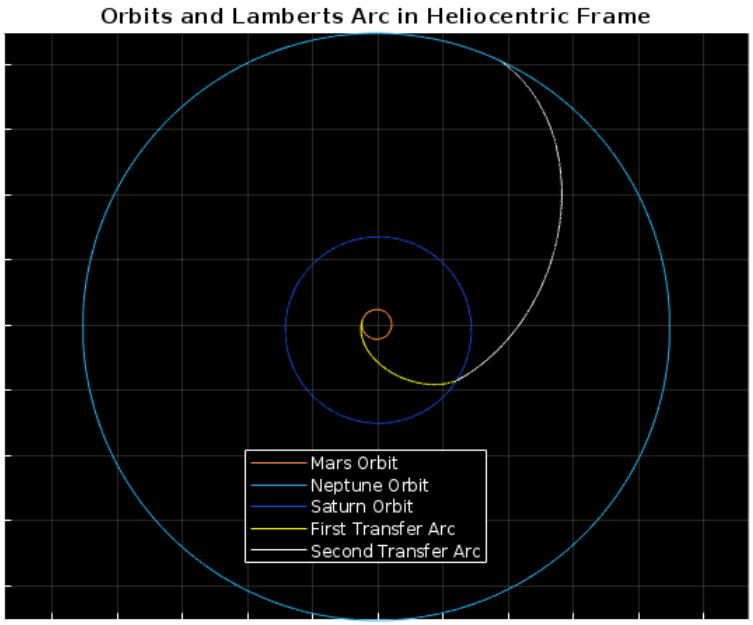
The minimum ΔV was found by using the genetic algorithm is $10.61\,\mathrm{km/s}$, so the transfer arcs are computed with respect to this ΔV transfer along with the v_{∞^+} and the v_{∞^-}:

Algorithm 2.4 Best Transfer Computation

```
% Compute bests transfer arcs
[~,~,~,~,VI_arc1,VF_arc1,~,~] = lambertMR(r1_arc,r2_arc,Dv_min_TOF_1,ksun);
[~,~,~,~,VI_arc2,VF_arc2,~,~] = lambertMR(r2_arc,r3_arc,Dv_min_TOF_2,ksun);
[rx_arc_1, ry_arc_1, rz_arc_1, vx_arc_1, vy_arc_1, vz_arc_1] = intARC_lamb(r1_arc,...
VI_arc1,ksun,Dv_min_TOF_1,86400);
[rx_arc_2, ry_arc_2, rz_arc_2, vx_arc_2, vy_arc_2, vz_arc_2] = intARC_lamb(r2_arc,...
VI_arc2,ksun,Dv_min_TOF_2,86400);

% V infinity
v_inf_min = (VF_arc1 - v_saturn');
v_inf_plus = (VI_arc2 - v_saturn');
```

In the plots below are shown the two correspondent Lambert's arcs in the heliocentric reference frame and in the planetocentric reference frame :



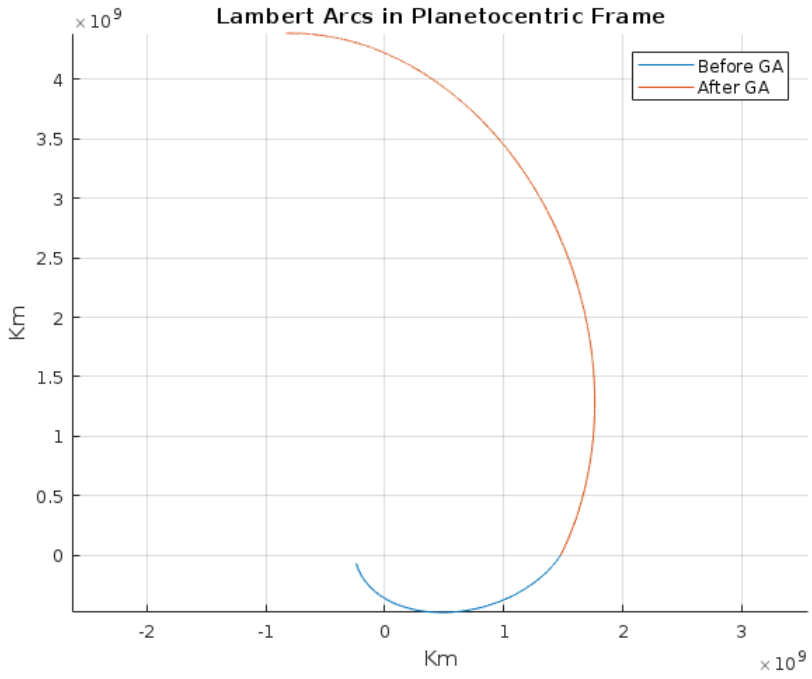


Figure 2.4: Lambert's Arcs Path

In order to compute the amount of ΔV that should be given at pericenter a constrained nonlinear system with boundary value conditions has to be solved to retrieve the radius of the pericenter, that can be then used to compute all the values needed to estimate the ΔV of the gravity assist. The system can be solved numerically using the MATLAB function **`fzero`**, setting as boundaries the planet's radius and the SOI radius. The function `powerflyby`, shown in algorithm 2.5, does the computation:

Algorithm 2.5 Powerflyby function

```
function dv = powerflyby(v_inf_min,v_inf_plus,k)

delta = acos(dot(v_inf_min,v_inf_plus)/(norm(v_inf_min)*norm(v_inf_plus)));

f = @(r_p) delta - asin(1./(1+(r_p*norm(v_inf_min)^2/k))) - asin(1./(1+(r_p*norm(v_inf_plus)^2/k)));
try
    r_p = fzero(f,[1e5 1e9]);
catch err
if (strcmp(err.identifier,'MATLAB:fzero:ValuesAtEndPtsSameSign'))
        dv = nan;
        return
    elseif (strcmp(err.identifier,'MATLAB:fzero:ValuesAtEndPtsComplexOrNotFinite'))
        dv = nan;
        return
    else
        rethrow(err)
    end
end

DELTA_min = r_p*sqrt(1 + 2*(k/(r_p*norm(v_inf_min)^2)));
DELTA_plus = r_p*sqrt(1 + 2*(k/(r_p*norm(v_inf_plus)^2)));

vp_min = (DELTA_min*norm(v_inf_min))/(r_p);
vp_plus = (DELTA_plus*norm(v_inf_plus))/(r_p);

dv = abs(vp_plus - vp_min);

end
```

where `delta` is the deviation angle δ. As result we get:

$$r_p = 1.27e+06\,\mathrm{km}$$

which means that the altitude from saturn is:

$$altitude = 1.21e+06\,\mathrm{km}$$

that is a reasonable result since the radius of Saturn's SOI is $5.45e+07$ km .
The parameters of the entering and exiting hyperbola can be calculated and are reported in table 2.1:

	Entering Hyperbola	Exiting Hyperbola
e	2.07187733056	2.07187733047
δ	0.96530811476 rad	0.96530811481 rad
Δ	$2.1588150264887e+06$ km	$2.1588150265496e+06$ km
θ_∞	2.07447891699 rad	2.0744789170 rad
β	1.06711373659 rad	1.06711373657 rad
v_p	9.55885975030 km/s	9.55885975015 km/s

Table 2.1: Hyperbolas Results

In the figure below is shown the flyby from a 2D and a 3D perspective

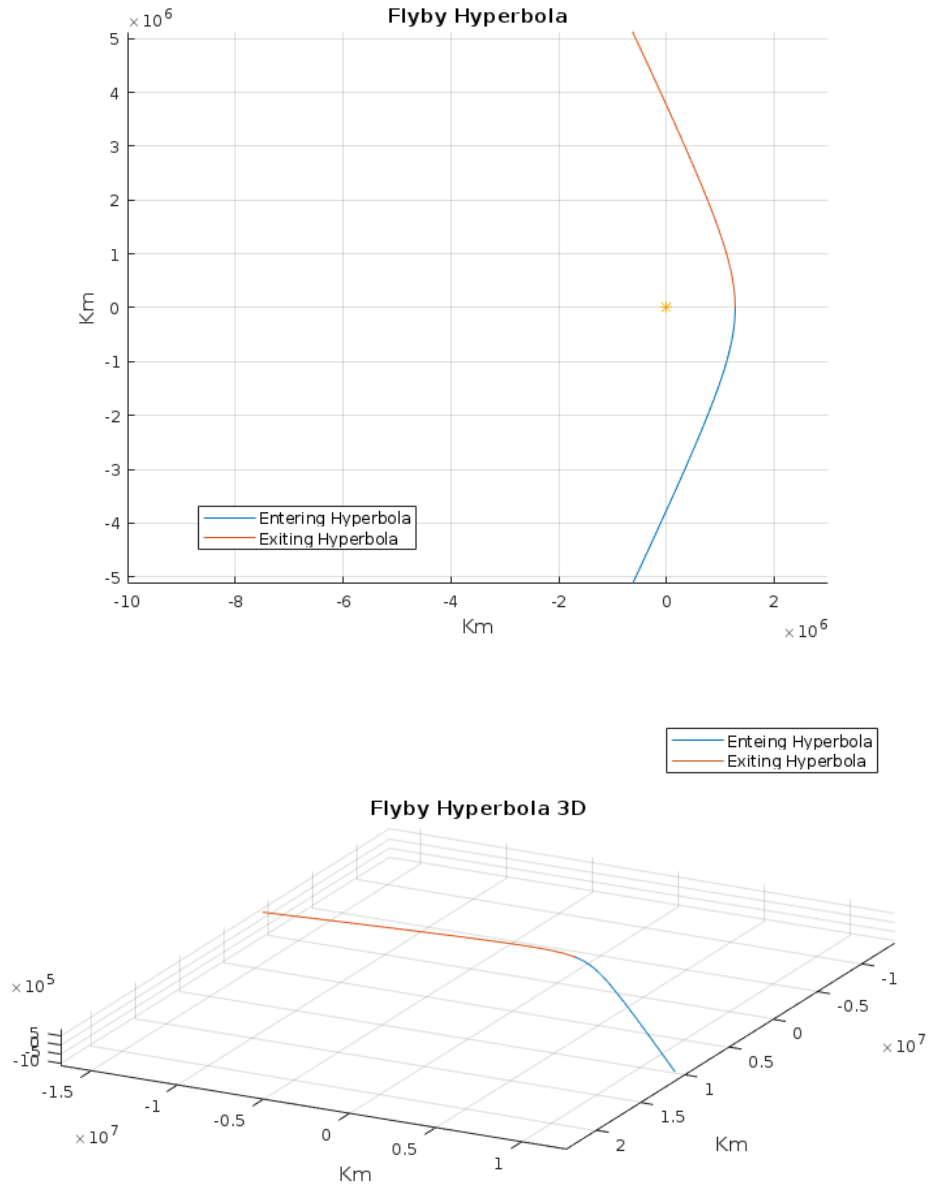


Figure 2.5: Hyperbolic Flyby

As we can see from the parameter of the two hyperbolas, a powered gravity assist for this transfer is nearly not needed, since those two orbits are almost lying on the same orbital plane.

The time needed for the flyby can be easily calculated from the time law by knowing the hyperbolic anomaly F:

Algorithm 2.6 Flyby Time

```
F_min = acosh((cos(theta_SOI_min) + e_min)/(1 + e_min*cos(theta_SOI_min)));
dt_min = sqrt(a_min^3/ksaturn)*(e_min*sinh(F_min)-F_min);
F_plus = acosh((cos(theta_SOI_plus) + e_plus)/(1 + e_plus*cos(theta_SOI_plus)));
dt_plus = sqrt(a_plus^3/ksaturn)*(e_plus*sinh(F_plus)-F_plus);
dt_tot = dt_min+dt_plus;
```

as a result we get a total flyby time of about 159.17 days, which is reasonable considering the large Saturn's SOI and the small total ΔV needed for the transfer.

Chapter 3

Orbital Perturbations

3.1 Introduction

The aim of this chapter is to explain the preliminary analysis of the propagation of orbit pertubations caused by the moon and the J2 perturbation of the orbit characterized by the following parameters:

Parameter	Value
a	$3.7067e4$ km
e	0.5682
i	0.8655 rad
Ω	$\pi/6$ rad
ω	$\pi/3$ rad
z_p	9627.33 km
J_2	$1.08e-3$

Table 3.1: Initial orbit

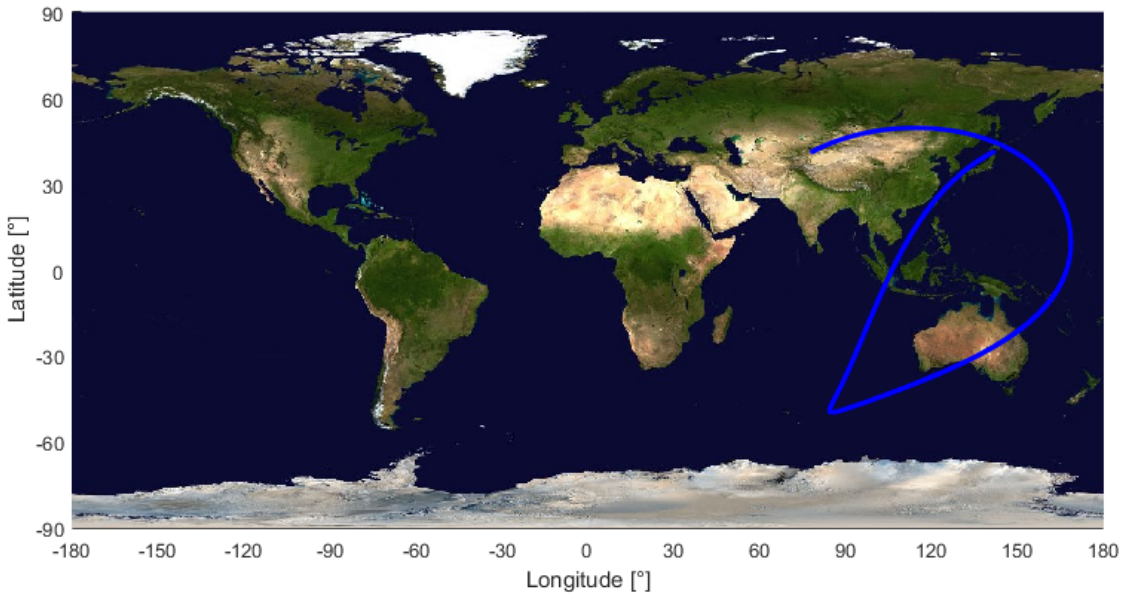


Figure 3.1: Ground track over one orbit

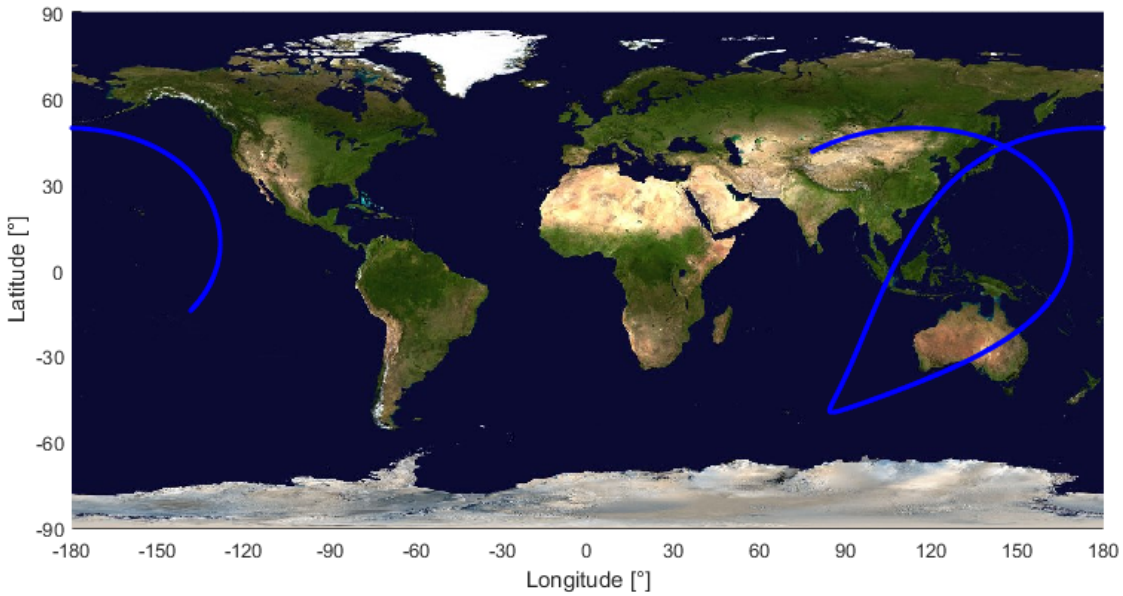


Figure 3.2: Ground track over one day

3.2 Perturbing acceleration

In our case the perturbing accelerastion$\mathbf{a}_p$ is given by:

$$\mathbf{a}_p = \mathbf{a}_{p_{J2}} + \mathbf{a}_{p_m}$$

where:

$$\mathbf{a}_{p_{J2}} = \frac{3}{2}\frac{J_2 \mu R^2}{r^4}\left[\frac{x}{r}\left(5\frac{z^2}{r^2}-1\right)\mathbf{\hat{\imath}} + \frac{y}{r}\left(5\frac{z^2}{r^2}-1\right)\mathbf{\hat{\jmath}} + \frac{z}{r}\left(5\frac{z^2}{r^2}-3\right)\mathbf{\hat{k}}\right]$$

$$\mathbf{a}_{p_m} = \mu_m\left(\frac{\mathbf{r}_{m/s}}{r_{m/s}^3} - \frac{\mathbf{r}_m}{r_m^3}\right)$$

3.3 The Gauss equations

Time evolution of perturbed orbital parameters can be estimated by means of the so called Gauss equation, that can be integrated with the `ode45` MATLAB function.

$$
\begin{cases}
\dfrac{da}{dt} = \dfrac{2a^2 v}{\mu} a_t \\
\dfrac{de}{dt} = \dfrac{1}{v}\left(2\left(e+\cos\theta\right)a_t - \dfrac{r}{a}\sin\theta a_n\right) & v = \sqrt{\dfrac{2\mu}{r} - \dfrac{\mu}{a}} & n = \sqrt{\mu/a^3} \\
\dfrac{di}{dt} = \dfrac{r\cos\theta^*}{h} a_h & b = a\sqrt{1-e^2} & h = nab \\
\dfrac{d\Omega}{dt} = \dfrac{r\sin\theta^*}{h\sin i} a_h & p = b^2/a & r = \dfrac{p}{1+e\cos\theta} \\
 & & \theta^* = \theta + \omega \\
\dfrac{d\omega}{dt} = \dfrac{1}{ev}\left(2\sin\theta a_t + \left(2e + \dfrac{r}{a}\cos\theta\right)a_n\right) - \dfrac{r\sin\theta^*\cos i}{h\sin i} a_h \\
\dfrac{d\theta}{dt} = \dfrac{h}{r^2} - \dfrac{1}{ev}\left(2\sin\theta a_t + \left(2e + \dfrac{r}{a}\cos\theta\right)a_n\right) \quad \text{or} \quad \dfrac{dM}{dt} = n - \dfrac{b}{eav}\left(2\left(1 + \dfrac{e^2 r}{p}\right)\sin\theta a_t + \dfrac{r}{a}\cos\theta a_n\right)
\end{cases}
$$

a_t, a_n, a_h are the components of the perturbing acceleration in the TNH reference frame, where the t component is parallel to the velocity, the h component is parallel to the angular momentum vector, and the n component is perpendicular to the other two.

A MATLAB function implementing those equation has been developed.

3.4 Direct dynamic integration

Another option that we have to find out the evolution of the perturbed problem is to directly insert into the 2 body equation the perturbing acceleration term, and then integrate the resulting equations:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} + \sum \mathbf{a}_p$$

3.5 Pertubations evolution

The variation of orbital parameter according to Gauss equation estimation and to the perturbed two body problem is shown in the plots below. In particular, in the semimajor axis plot the periodic motion of the orbit can be seen by means of the highest peaks of the plot:

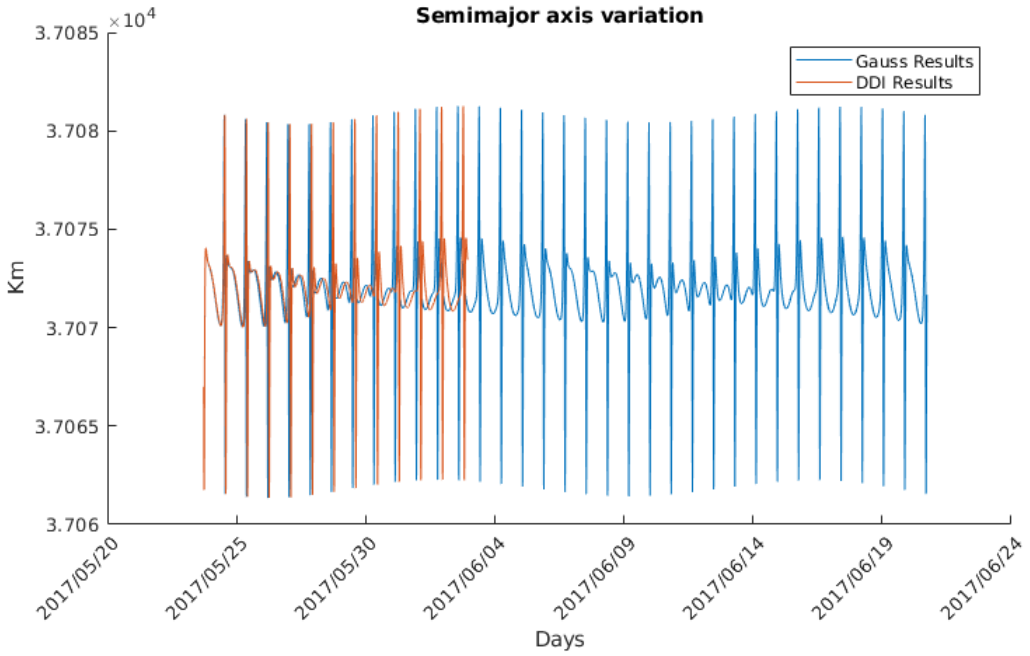


Figure 3.3: Semimajor Axis

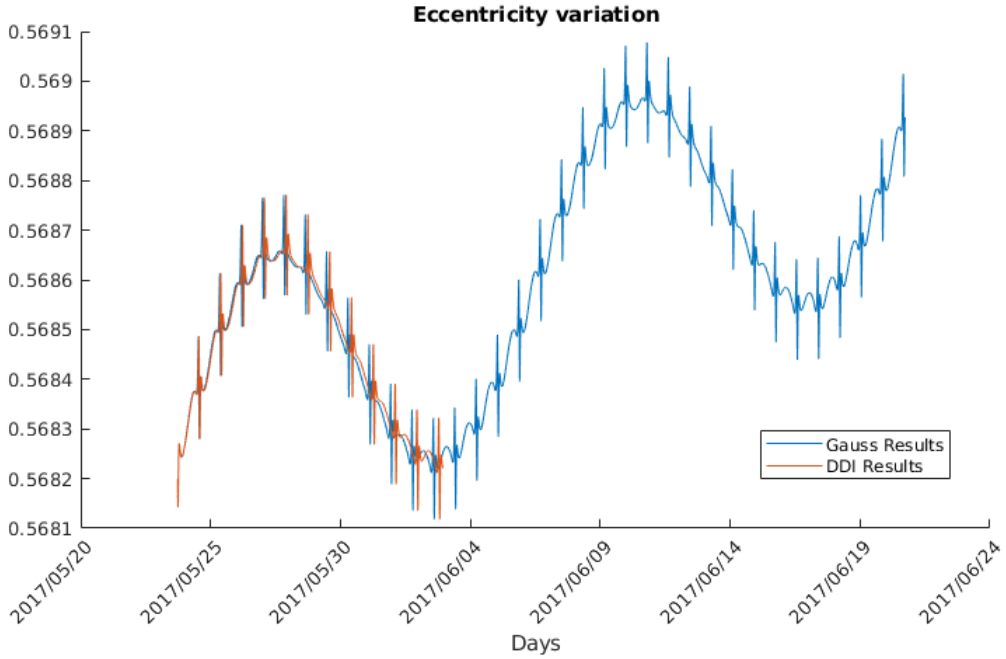


Figure 3.4: Eccentricity

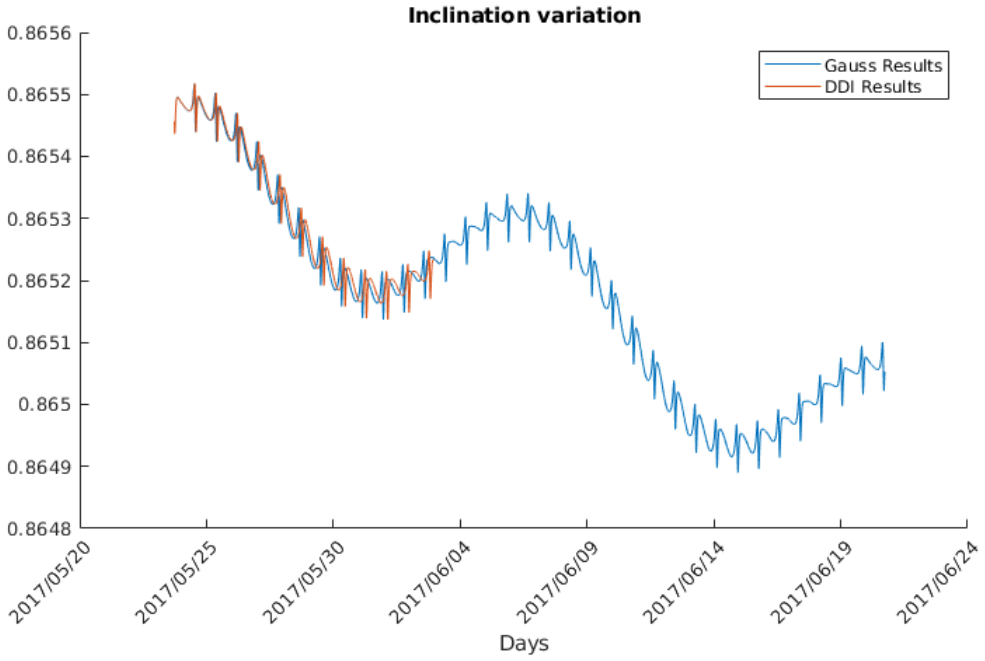


Figure 3.5: Inclination

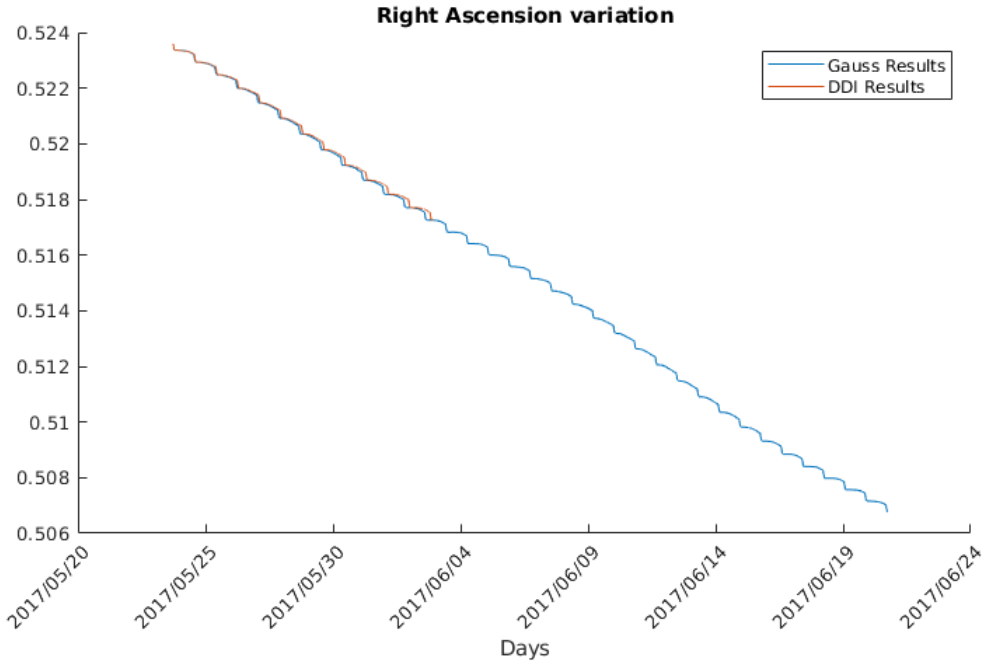


Figure 3.6: Right Ascension

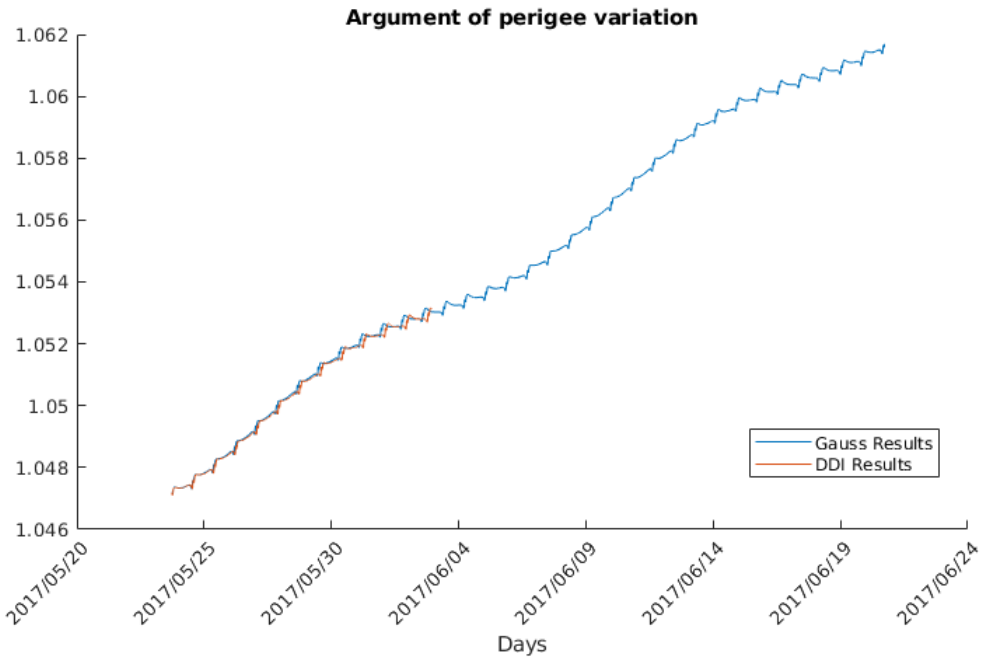


Figure 3.7: Argument of perigee

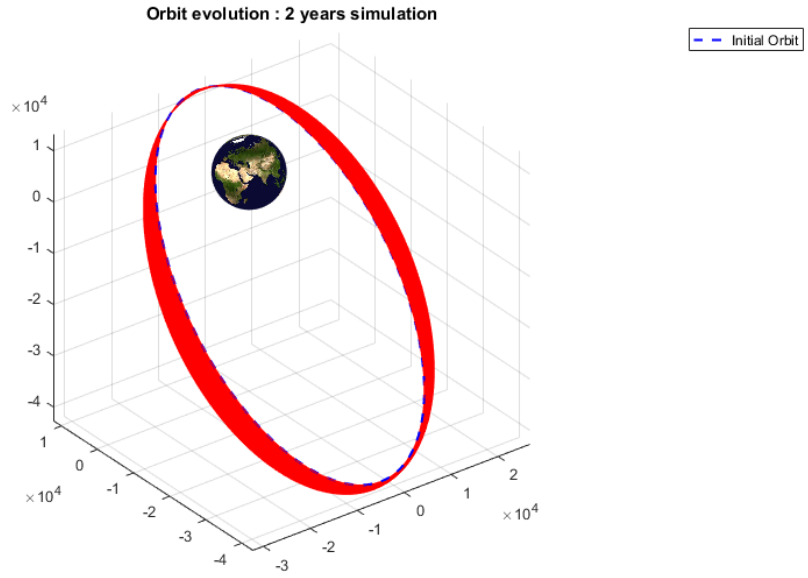


Figure 3.8: Orbit evolution

As we can see the Gauss equation represents a good approximation of the perturbed motion.

3.6 Secular component

We can put in evidence the secular trend of the perturbations by cutting of all the high frequency components of the orbital pertubation with a low pass filter.

In order to find out the cutoff frequency for the low pass filter, a frequency spectrum analysis has been done on the orbital parameters obtaind by the integration.

The frequency content associated with each parameter con be seen in the figures below:

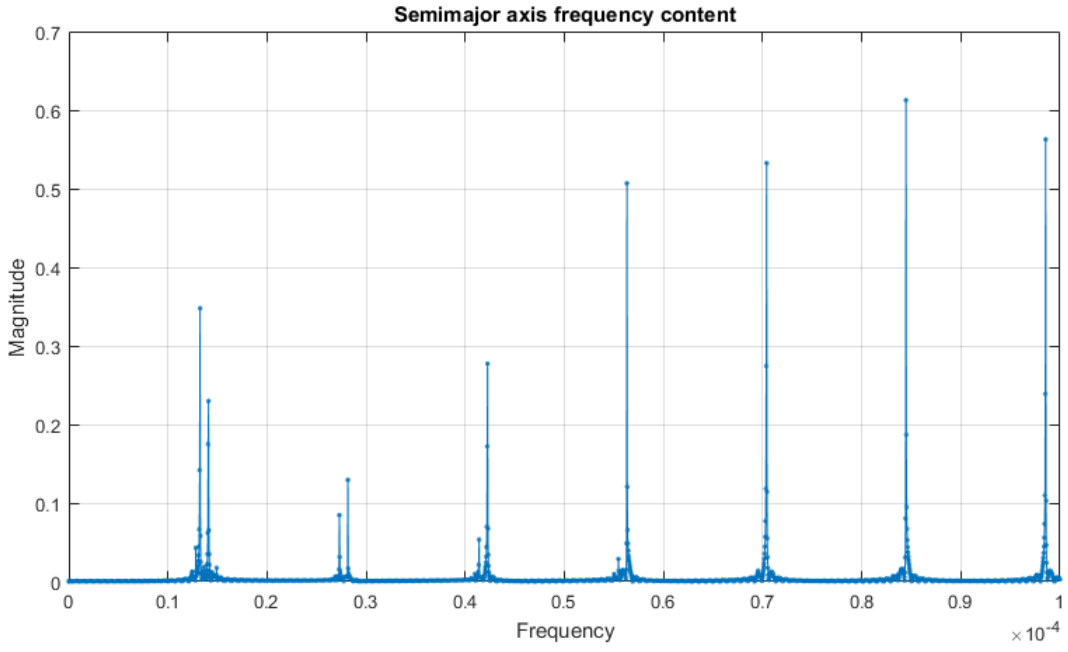


Figure 3.9: Semimajor Axis

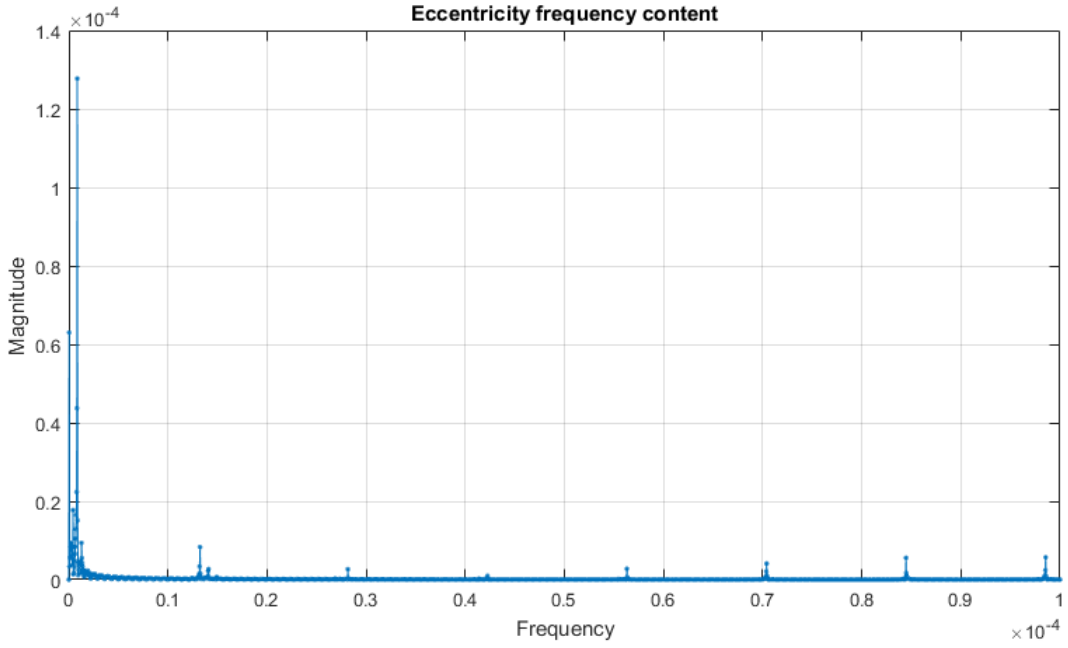


Figure 3.10: Eccentricity

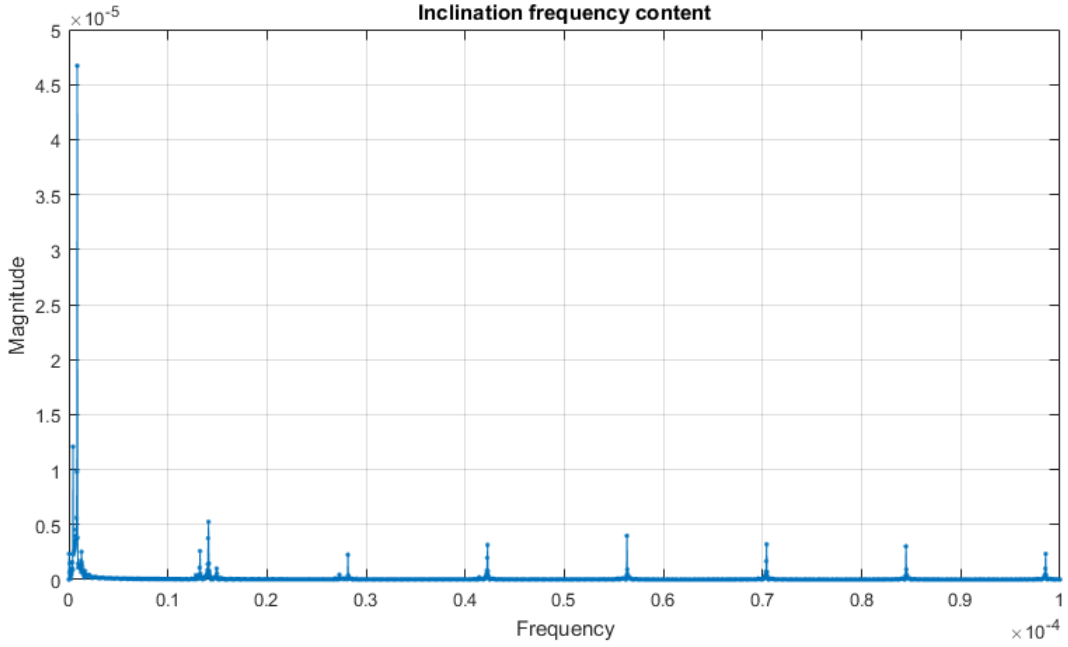


Figure 3.11: Inclination

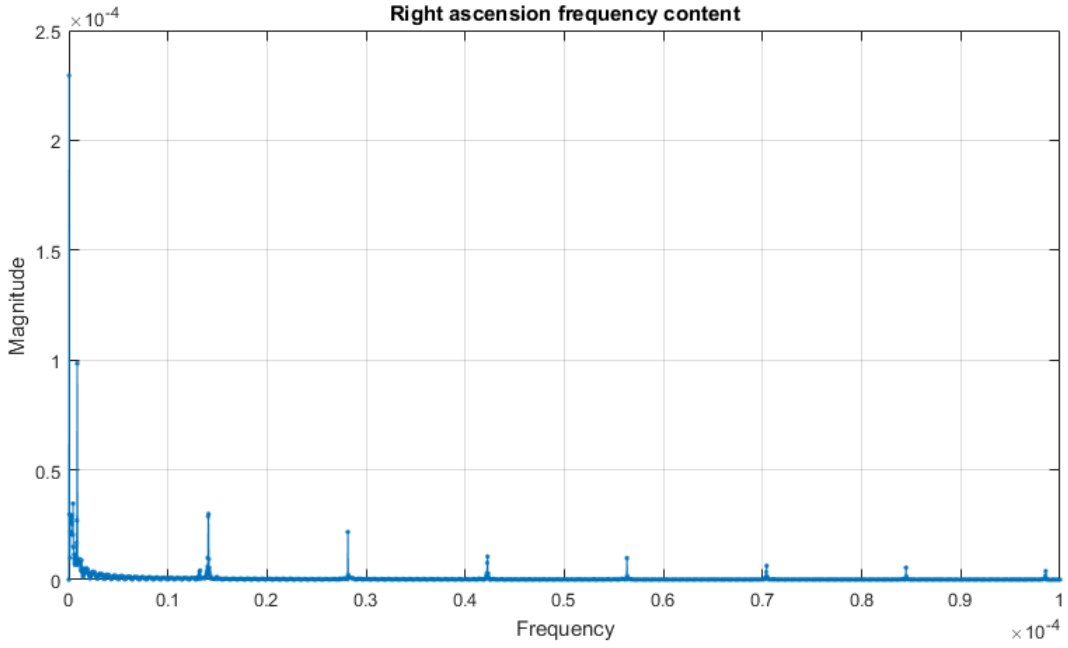


Figure 3.12: Right Ascension

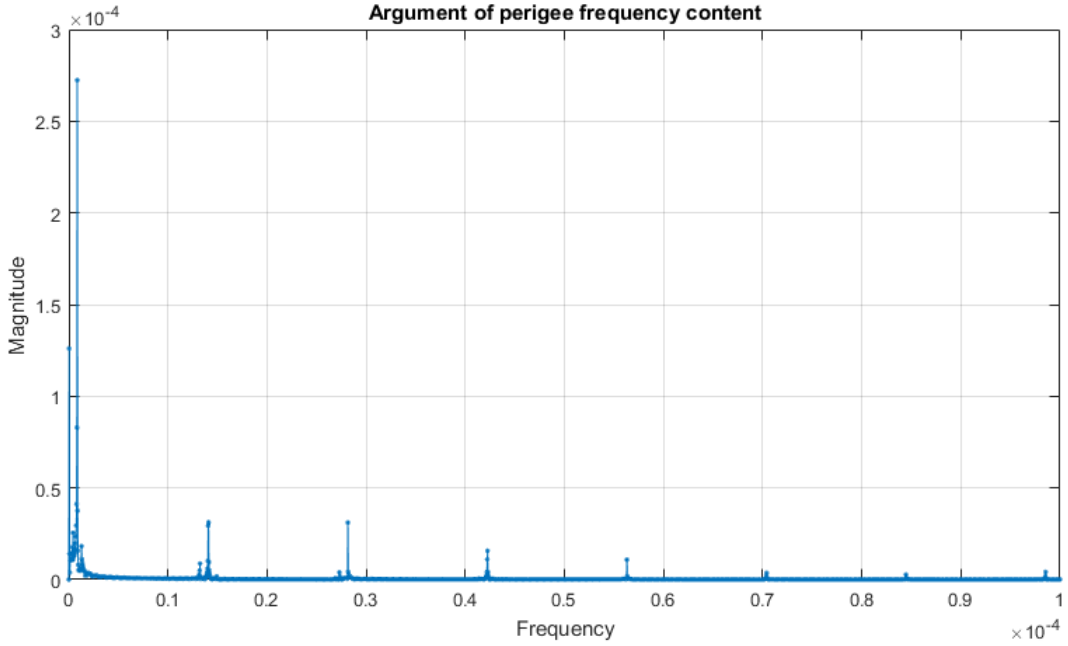


Figure 3.13: Argument of perigee

It should come with no surprise that the first frequency of the semimajor axis perturbation frequency content is

the one related to the orbital period, because it represents the fastest “dynamic” of the orbital motion. It is also present in the other parameters’ perturbation frequency content, though it is not the first one.

By choosing half of the inverse of the orbital period as cutoff frequency for a second order Butterworth lowpass filter, we can obtain the secular trend of the orbital parameters.

Concerning the semimajor axis variation, it can be observed that it is variating around the mean value of abount some tens of meters, so it is almost constant. It can also be observed the presence af a very low frequency oscillation, which is related to the moon orbital period, as it completes two periods every sideral month.

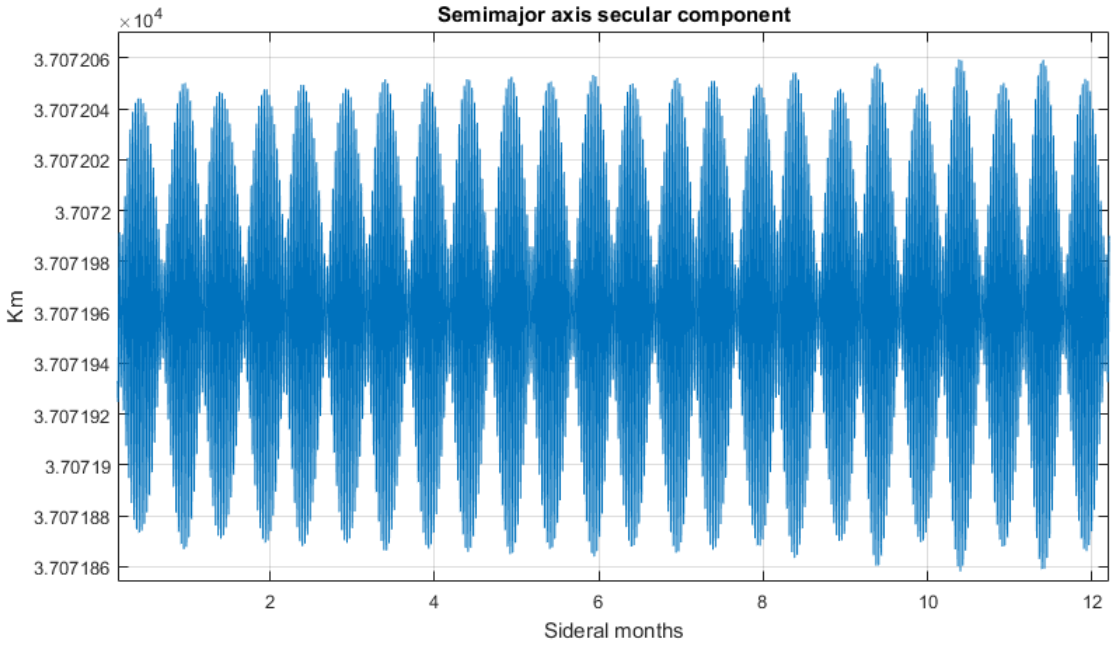


Figure 3.14: Semimajor axis

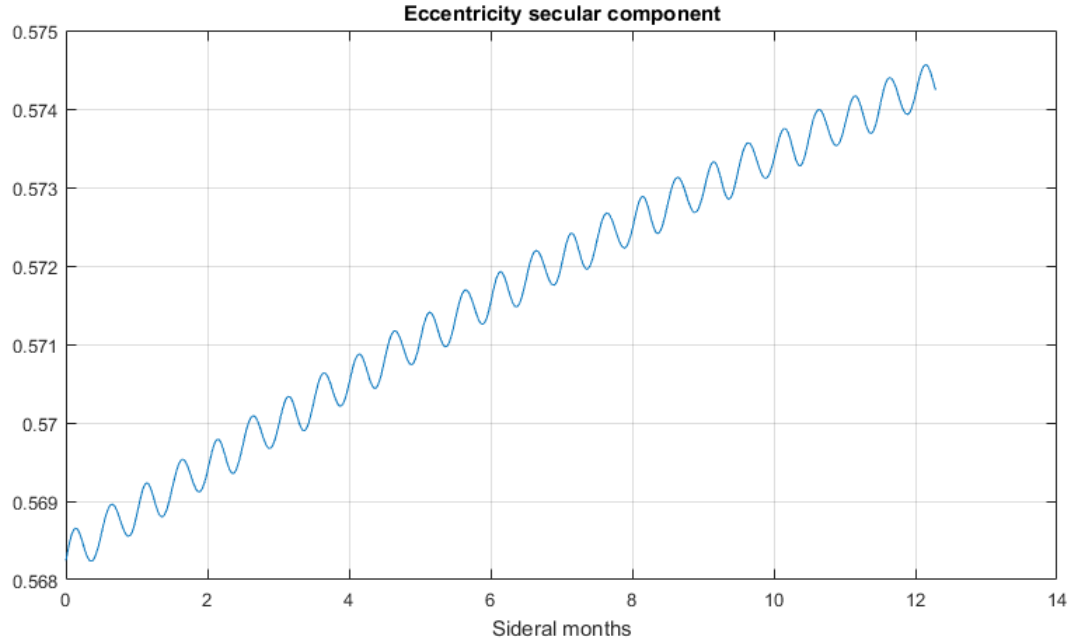


Figure 3.15: Eccentricity

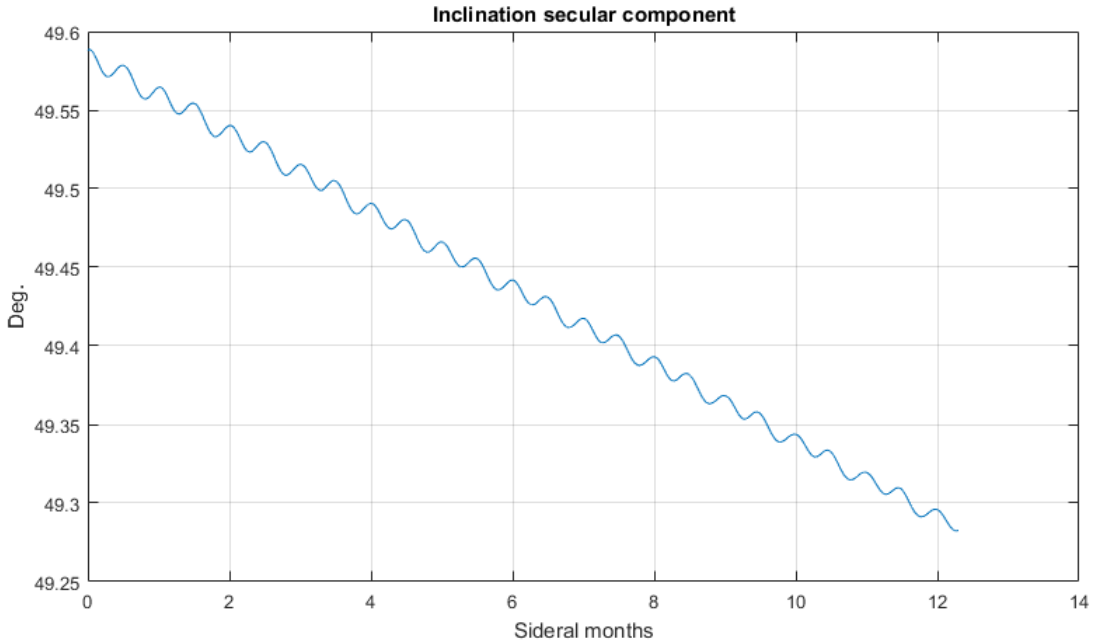


Figure 3.16: Inclination

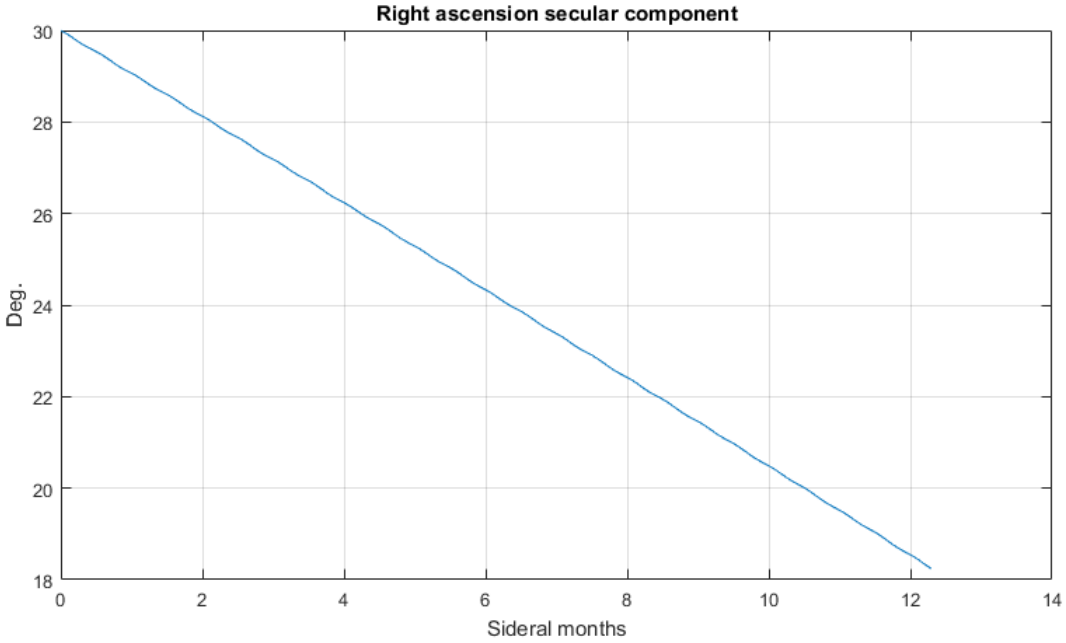


Figure 3.17: Right Ascension

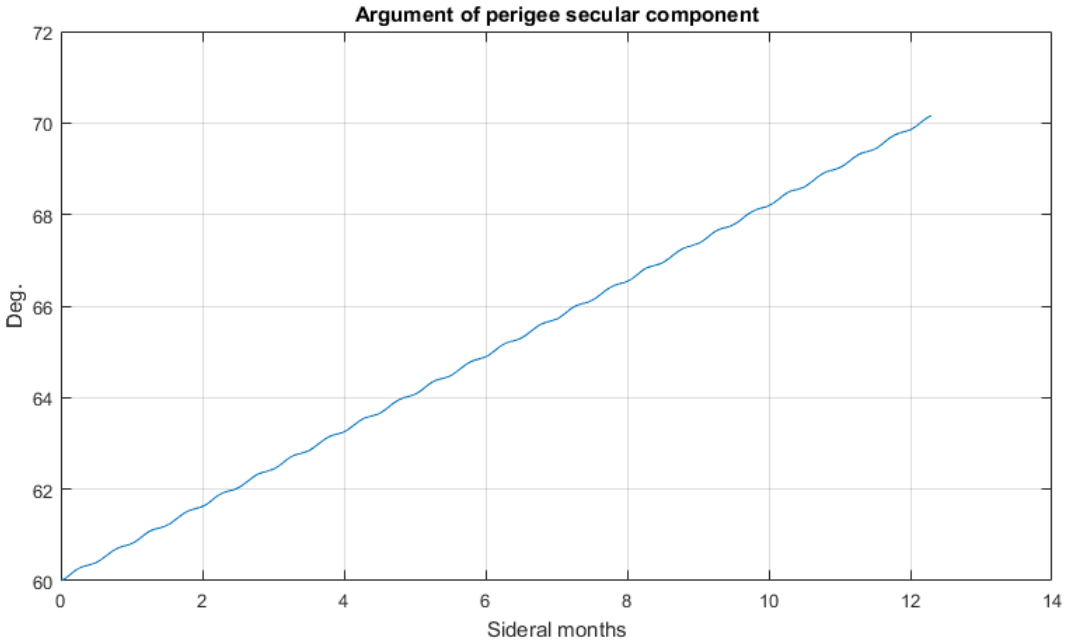


Figure 3.18: Argument of pericenter