OITS Documentation

Adam Hibberd

Contents

i) Reference Documentation

i) Optimum Interplanetary Trajectory Software 2017 Definition File (Theory Doc)

ii) OITS User Guide

iii) Fundamentals of Astrodynamics, Mueller, Bate & White

Optimum Interplanetary Trajectory Software

(OITS)

1) Static Overview

1.1) Introduction

Optimum Interplanetary Trajectory Software (OITS) is a software application developed by Adam Hibberd, originally using the MATLAB Integrated Development Environment (IDE) and exploiting the MALTAB programming language. In principle it should work on any platform/operating system which supports MATLAB. It is executed by typing *run_OITS* at the MATLAB command line (but first read Section (1.2) OITS Dependencies).

The principal purpose of OITS is preliminary interplanetary spacecraft mission design and interplanetary mission feasibility studies. The trajectories may incorporate one or more gravitational assists (GA's). The theory adopts various assumptions, including that of a patched conic. The theory document is available at ref (i).

It must be reinforced here that this software is for preliminary research only and cannot be used as a comprehensive means of defining and analysing a specific mission. Such software, where it exists, would take into account other forces, such as non-gravitational, and particular spacecraft design features, as well as chosen launch vehicle characteristics.

Nevertheless the assumptions made are valid for most mission scenarios, as is attested by the fidelity of solutions obtained by OITS with respect to real historical missions.

Table 1.1.1 shows a comparison of the results generated by OITS for two missions, the Mars Insight Lander mission (direct from Earth to Mars with launch in 2018) and New Horizons mission (from Earth to flyby Pluto with a Gravitational Assist, GA, of Jupiter).

The interplanetary trajectories computed assume the sun alone is the gravitational influence on the spacecraft, and further that the force follows the law of gravitation discovered by Newton, which can be expressed as follows:

$$\ddot{\underline{r}} = -\frac{\mu}{r^3}\underline{r} \qquad (1)$$

- $\underline{\mathbf{r}}$ is the position of the body with respect to the centre of the solar system
- μ is the gravitational mass of the sun = $1.32712440018 \times 10^{20}$ m^3s^{-2}
- $r = |\underline{\mathbf{r}}| = \text{norm}(\underline{\mathbf{r}})$

Table 1.1.1 Comparison of OITS Results with Two Real Missions: Mars Insight and New Horizons

NASA Mission	OITS Trajectory	Launch Year	Actual Launch Date	OITS Launch Date	Actual Arrival Date	OITS Arrival Date	Actual Overall Mission Duration (days)	OITS Overall Mission Duration (days)	Actual Date of Jupiter Closest Approach	OITS Date of Jupiter Closest Approach	Actual Altitude at Closest Approach to Jupiter (km)	OITS Altitude at Closest Approach to Jupiter (km)
Mars Insight Lander	Earth-Mars	2018	5th May	11th May	26th Nov 2018	2nd Dec 2018	205	205	N/A	N/A	N/A	N/A
New Horizons	Earth-Jupiter-Pluto	2006	19th Jan	20th Jan	14th Jul 2015	14th Jul 2015	3463	3462	28th Feb 2007	1st Mar 2007	2.3 million	2.3 million

At each planetary 'node' between successive interplanetary trajectories, as the spacecraft encounters the planet, the gravity of that planet is assumed to dominate, and that of the sun is assumed negligible. This amounts to a 'patched conic' approximation.

One of the variables used by OITS is *Nbody*, denoting the number of Solar System Objects (either planets, asteroids, comets or otherwise) which the user has specified, inclusive of the terminal encounters, for the mission he wishes to investigate. Hence from Table 1.1.1, we have for Mars Insight Lander, *Nbody=2*, and for New Horizons, *Nbody=3*.

For the JUpiter Icy Moon Explorer to be launched in 2022, the sequence is Earth, Earth, Venus, Earth, Mars, Earth, Jupiter, a total of *Nbody=7* (note the double Earth-Earth sequence is considered as two separate encounters). This JUICE mission will be adopted in future sections as an illustration to elucidate the operation of OITS.

Within OITS, there are six levels of functionality required to calculate an Optimum Trajectory, refer Table 1.1.2.

Table 1.1.2

Number	Functionality	Requirements	Resources
(1)	**Specification** Mission & Optimizer	User inputs	OITS GUI
(2)	**Compute SSO position and velocity**	Binary Spice Kernel files (.bsp) Times(i),i=1:Nbody	NASA JPL SPICE Toolkit
(3)	**Compute Interplanetary Trajectories**	Outputs of (2)	OITS/ *Transfer_orbit.m*
(4)	**Compute Encounter Trajectories** at each SSO	Outputs of (3) Nbody>2	OITS/ *Connecting_Hyperbola.m*
(5)	**Optimization** of Trajectory	Outputs of (1), (2), (3) & (4) Times(i)	Nonlinear Global Optimization Tool (NOMAD)
(6)	**Results Analysis**	Results of (5)	OITS

1.2) OITS Dependencies

1.2.1) Required Directory Structure and Files

Figure 1.2.1.1 shows the required directory and file structure for running OITS. **Note that for the github repository there is an extra directory, '*thirdparty*', between the root layer '*OITS*', and the subdirectories '*SPICE*', '*Opti-master*' & '*NLOPT*'.**

The following lists the activities which must be performed before OITS can be run:

I. The SPICE directory in Figure 1.2.1 must be created and then the NASA JPL SPICE toolkit must be downloaded and installed into it, from the following link (this is unnecessary for the github version): https://naif.jpl.nasa.gov/naif/toolkit_MATLAB.html

II. The NASA JPL Spice binary kernel file de430.bsp should be inserted in the SPICE directory shown in Figure 1.2.1 and can be found at the following link: https://naif.jpl.nasa.gov/pub/naif/pds/wgc/kernels/spk/

III. The NASA JPL leap second file, naif0012.tls, should be present in the SPICE directory and can be found at the following link (this is already the case for the github version): https://naif.jpl.nasa.gov/pub/naif/pds/wgc/kernels/lsk/

IV. The directory Opti-master can be found at (supplied with the github version): https://github.com/jonathancurrie/OPTI

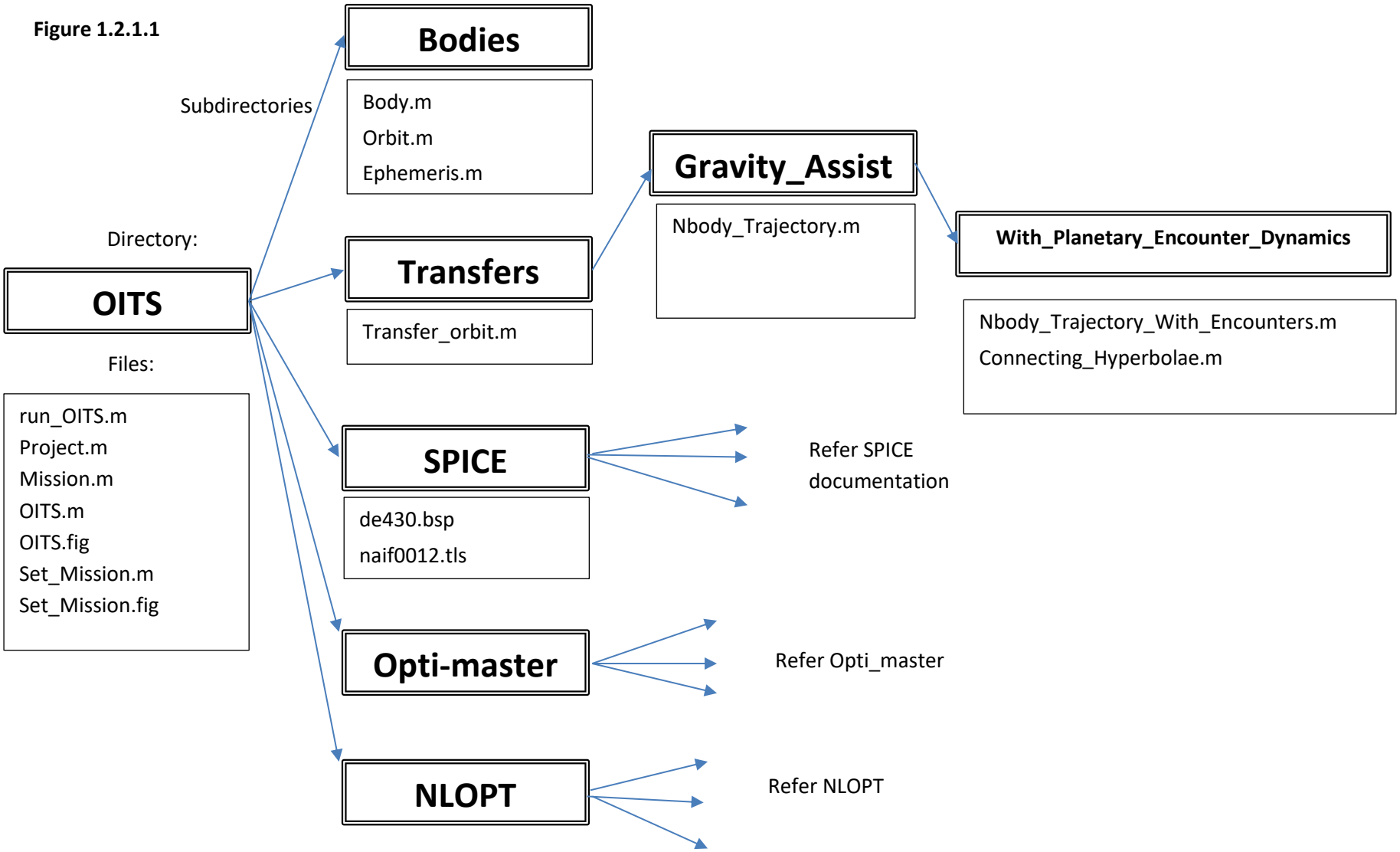
V. Before running OITS, execute opti_Install.m in folder OPTI-master. This generates various solvers accessed by OITS, in particular NOMAD

VI. Alternative Non-Linear Solvers can be found here: https://nlopt.readthedocs.io/en/latest/#nlopt_1

VII. The directory NLOPT is generated as a result of installing VI.

VIII. To MEX the NLOPT files from VII, The MinGW-w64 C++ compiler must be installed and instructions can be found here for doing this (this MEX is automatically performed by run_OITS.m): https://uk.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler

Figure 1.2.1.1


Bodies
Subdirectories
Body.m
Orbit.m
Ephemeris.m
Gravity_Assist
Nbody_Trajectory.m
With_Planetary_Encounter_Dynamics
Nbody_Trajectory_With_Encounters.m
Connecting_Hyperbolae.m
Directory:
OITS
Transfers
Transfer_orbit.m
Files:
run_OITS.m
Project.m
Mission.m
OITS.m
OITS.fig
Set_Mission.m
Set_Mission.fig
SPICE
de430.bsp
naif0012.tls
Refer SPICE documentation
Opti-master
Refer Opti_master
NLOPT
Refer NLOPT

1.2.2) NASA SPICE toolkit & Binary SPICE Kernel Files

As outlined in Section (1.1), to calculate accurate ephemerides of a particular SSO to be studied as a target of an interplanetary mission, the corresponding binary SPICE Kernel file (*.bsp) should be generated by the user via NASA's Horizons telnet access portal. This can be found here:

https://ssd.jpl.nasa.gov/?horizons

The corresponding instructions should be followed and the *.bsp file can be downloaded from the ssd.jpl.nasa.gov server using an ftp session. A typical ftp session may look as follows:

- `o ssd.jpl.nasa.gov`
- `username: ftp`
- `password: user's email address`
- `cd /pub/ssd/`
- `binary`
- `get wld2345.15 C:\Users\fred\Downloads\newname.bsp`
- `close`
- `quit`

Having downloaded the file newname.bsp using the above process, this file must be moved or copied into the SPICE subdirectory of OITS (see Figure 1.2.1.1). It is finally a question of including a reference to the newname.bsp file in the MATLAB script run_OITS.m as show in Figure 1.2.2.1

Figure 1.2.2.1 Necessary mods to file run_OITS.m

```
This = This.Initialize_SPICE;

This = This.Get_SPICE_List(This.BSP);
This = This.Add_Intermediate_Point;
This = This.Add_Fixed_Point;
This = This.Add_Custom_Body;
This = This.Get_SPICE_List('SPICE\OUMUAMUA.bsp');
This = This.Get_SPICE_List('SPICE\halley.bsp');
This = This.Get_SPICE_List('SPICE\Bennu.bsp');
This = This.Get_SPICE_List('SPICE\newname.bsp');
This = This.Merge_Body_Data;
```

By default, the file de430.bsp must be present in the SPICE directory. This file is pre-generated by NASA and can be downloaded from:

https://naif.jpl.nasa.gov/pub/naif/pds/wgc/kernels/spk/

Also the leap-second file naif0012.tls must be present and can be downloaded from:

https://naif.jpl.nasa.gov/pub/naif/pds/wgc/kernels/lsk/

The de430.bsp file contains ephemeris data for the following planets. The planets covered by de430.bsp are summarised in Table 1.2.2.1

Table 1.2.2.1

PLANET	SPICE ID	SPICE Time Interval
MERCURY BARYCENTER	1	1549 DEC 31 (TDB) to 2650 JAN 25 (TDB)
VENUS BARYCENTER	2	As above
EARTH BARYCENTER	3	As above
MARS BARYCENTER	4	As above
JUPITER BARYCENTER	5	As above
SATURN BARYCENTER	6	As above
URANUS BARYCENTER	7	As above
NEPTUNE BARYCENTER	8	As above
PLUTO BARYCENTER	9	As above
SUN	10	As above
MERCURY	199	As above
VENUS	299	As above
MOON	301	As above
EARTH	399	As above

1.3) Structure of OITS

Table 1.3.1 lists all the classes defined by OITS and their purposes.

Table 1.3.1

Class	Purpose
Project	Overall Project Spec.
Mission	Trajectory Declaration and ΔV Output
Body	Encounter Body Spec.
Ephemeris	Ephemerides/Position & Velocity
Orbit	Keplerian Orbit Spec.
Transfer_orbit	Interplanetary Orbit Calculation
Nbody_Trajectory	Trajectory Computation without Encounter Dynamics
Nbody_Trajectory_With_Encounters	Trajectory Computation with Encounter Dynamics (subclass of Nbody_Trajectory)
Connecting_Hyperbolae	Encounter Hyperbolae

The script file to execute OITS is *run_OITS.m*.

The primary purpose of *run_OITS.m* is to declare a new *Project* object called *This*.

This is then modified by the user by employing a GUI which is called from *run_OITS.m*. This GUI is managed by *OITS.m* and the layout of the introductory menu for the GUI is specified in *OITS.fig*.

The general structure of OITS is shown in Figure 1.3.1.

The attribute *Solution* of *Project* is the most recent optimization of the current mission, the latter being specified by *Current_Mission* (also an attribute of *Project*). *Solution* is updated by the method *Optimize_Mission* of *Project* wherein the Nonlinear Programming Solver is called (default is NOMAD) to solve the problem presented to it by various *Project* attributes including *Current_Mission*. Once *Solution* has been updated by *Optimize_Mission*, it is copied to *Current_Mission* as follows:

```
obj.Current_Mission = obj.Solution;
```

There is only one example of class inheritance in OITS, see Figure 1.3.2.

Figure 1.3.1

General Structure of OITS

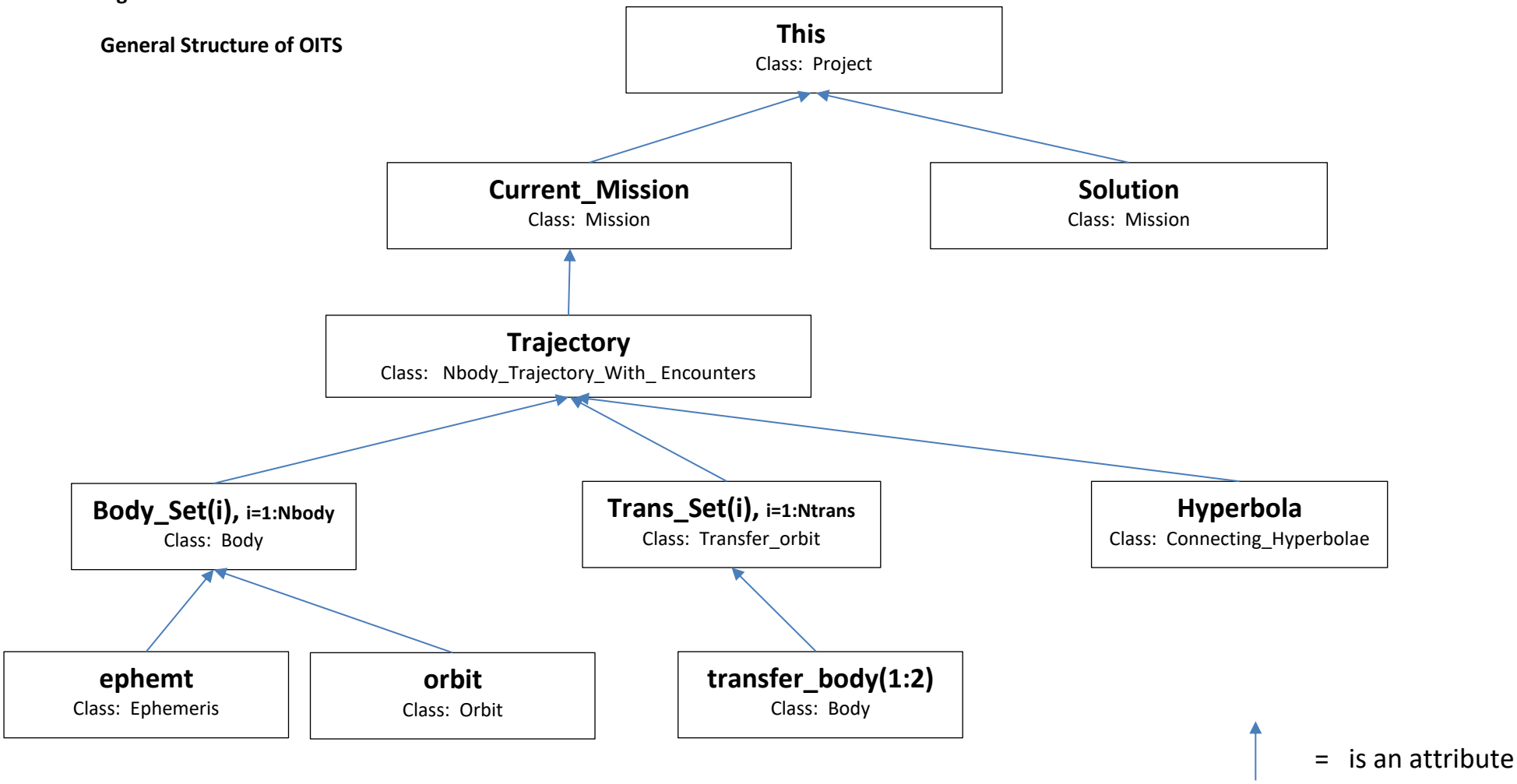
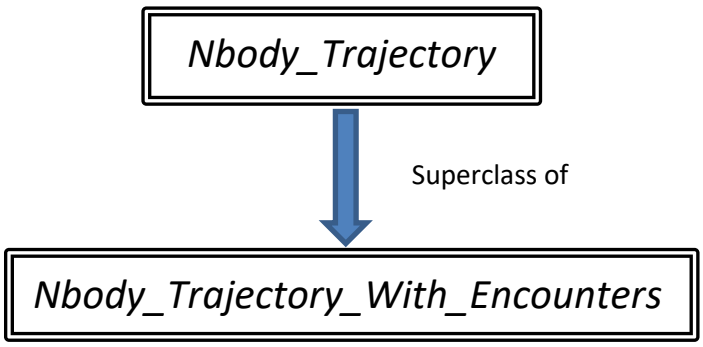


Figure 1.3.2



1.3.1) Outline of *Project* Attribute *Solution* and its Attribute *Trajectory*

OITS is activated by executing the *run_OITS.m* file wherein a global variable is declared, *This* which is of type *Project*. All the trajectory design, optimization and analysis is performed on *Project This*. In addition an attribute of *Project This*, i.e. *Current_Mission*, which is of class *Mission* needs to be populated by various user-specified data pertaining to the interplanetary trajectory. An attribute of *Mission* is *Trajectory* which is of class *Nbody_Trajectory_With_Encounters* and holds the current trajectory profile, which is a function of the encounter times of the spacecraft at each of the bodies visited, and possibly also other information, for example if one of the bodies is an 'Intermediate Point'.

The Optimization process is performed by *This.Optimize_Mission()* on *This.Solution*, which is a copy of *This.Current_Mission*. Its purpose is to find the optimal values of *This.Solution* and specifically its attribute *This.Solution.Trajectory*.

For the purposes of explanation, from hereon we shall adopt an example mission in the form of the ESA mission JUpiter Icy Moon Explore (JUICE) to Jupiter. With a launch in 2022, the planetary sequence is as follows: Earth, Earth, Venus, Earth, Mars, Earth, Jupiter. Thus we have the number of encounters as 7:

```
Solution.Trajectory.Nbody = 7
```

Furthermore there are 6 transfer trajectories, because there are *Nbody-1* consecutive pairs of encounters:

```
Solution.Trajectory.Ntrans = 6
```

Refer to Figure 1.3.1.1 for the main attributes of *Trajectory* for the JUICE mission.

There are also two further important attributes of *Solution* which are related these are:

```
Solution.Mission_Times(i)   i=1:Solution.Trajectory.Nbody

Solution.Absolute_Times(i)  i=1:Solution.Trajectory.Nbody
```

Absolute_Times is an array of absolute encounter times for each of the bodies on the left-hand side of Figure 1.3.1.1 in turn. **The times are TBD (Barycentric Dynamical Time) in seconds.**

Mission_Times is an array of encounter times for each of the Bodies on the left-hand side of Figure 1.3.1.1 in turn. In particular the first element, M*ission_Times(1)* is the absolute time, TBD (Barycentric Dynamical Time) in seconds. Subsequent elements are Times-of-Flight (TOF), between successive encounters.

Thus for example

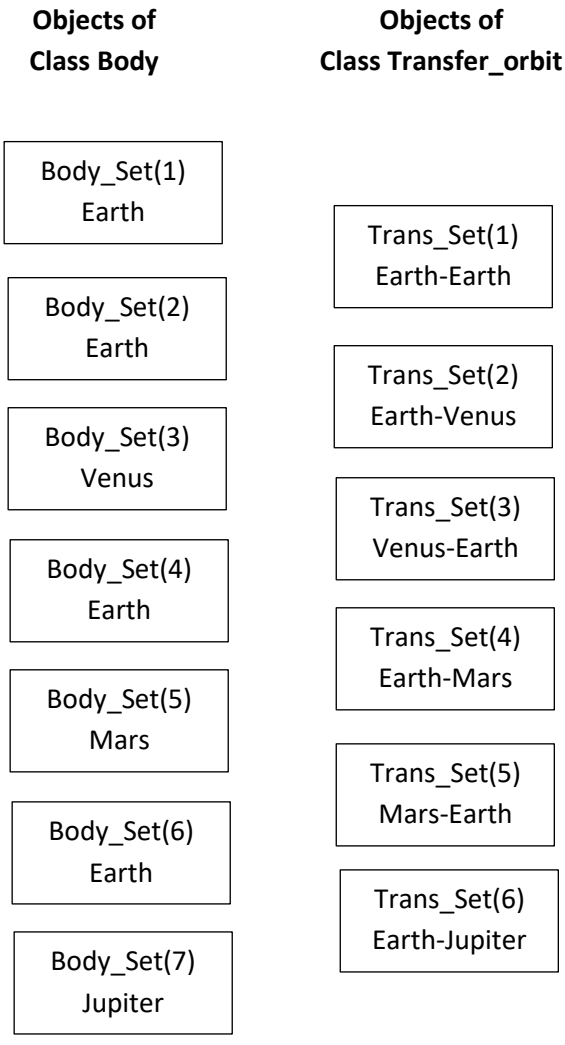
```
Solution.Mission_Times(4)=Solution.Absolute_Times(4)-Solution.Absolute_Times(3)
```

but

```
Solution.Mission_Times(1)=Solution.Absolute_Times(1)
```

Figure 1.3.1.1

Main Attributes of *Solution.Trajectory* for the ESA JUICE Mission



It is the purpose of *This.Optimize_Mission()*, method of *Project*, to find the optimal values of *Mission_Times*. Note that *Absolute_Times* can be derived directly from *Mission_Times*, and vice versa.

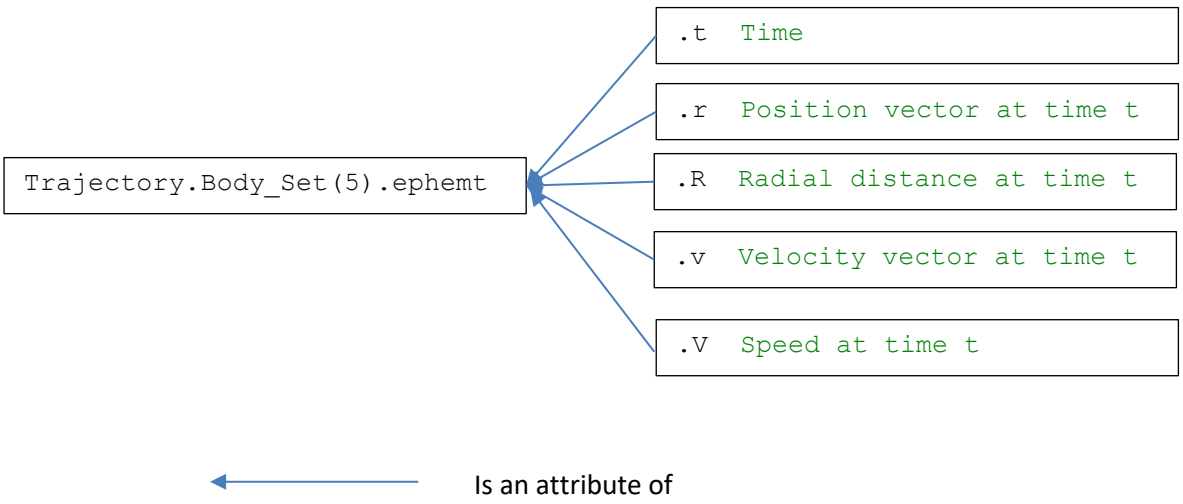
We can now analyse further the main attributes of *Solution.Trajectory* in Figure 1.3.1.1.

1.3.2) Outline of *Trajectory* Attribute *Body_Set*

Continuing with the theme of JUICE (see previous section), take an arbitrary body along the interplanetary trajectory, let's say *Trajectory.Body_Set(5)* which is *Mars*. This will have an ephemeris at time t of *Trajectory.Body_Set(5).ephemt*, as is shown in Figure 1.3.2.1

NOTE all ephemerides are in the NASA SPICE reference frame 'ECLIPJ2000', the Earth mean ecliptic and equinox of the epoch J2000.

Figure 1.3.2.1



Furthermore there is the *orbit* attribute of *Body_Set(5)* which is of class *Orbit* and this is shown in Figure 1.3.2.2.

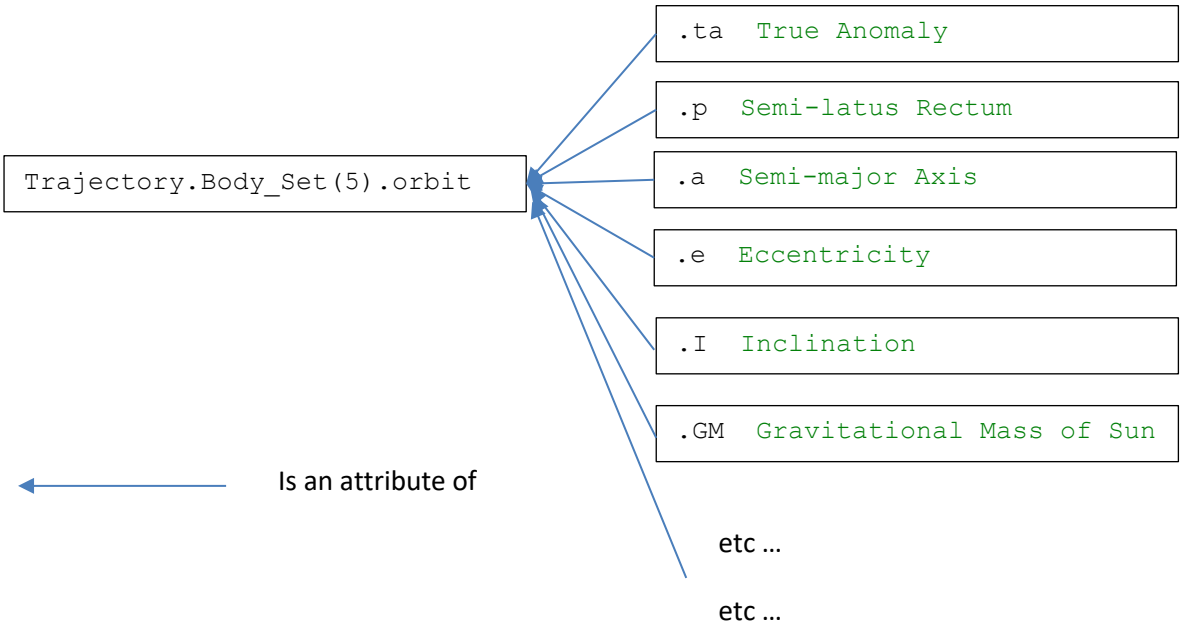
There are various other attributes of *Body_Set(5)*, including a flag indicating whether the object is actually an *Intermediate Point*, or it is a *Custom Body* as follows:

```
Trajectory.Body_Set(5).Fixed_Point =1  ! Object is an Intermediate point

Trajectory.Body_Set(5).Fixed_Point =0  ! Object is not an IP

Trajectory.Body_Set(5).Fixed_Point =-1 ! Object is a Custom Body
```

Figure 1.3.2.2

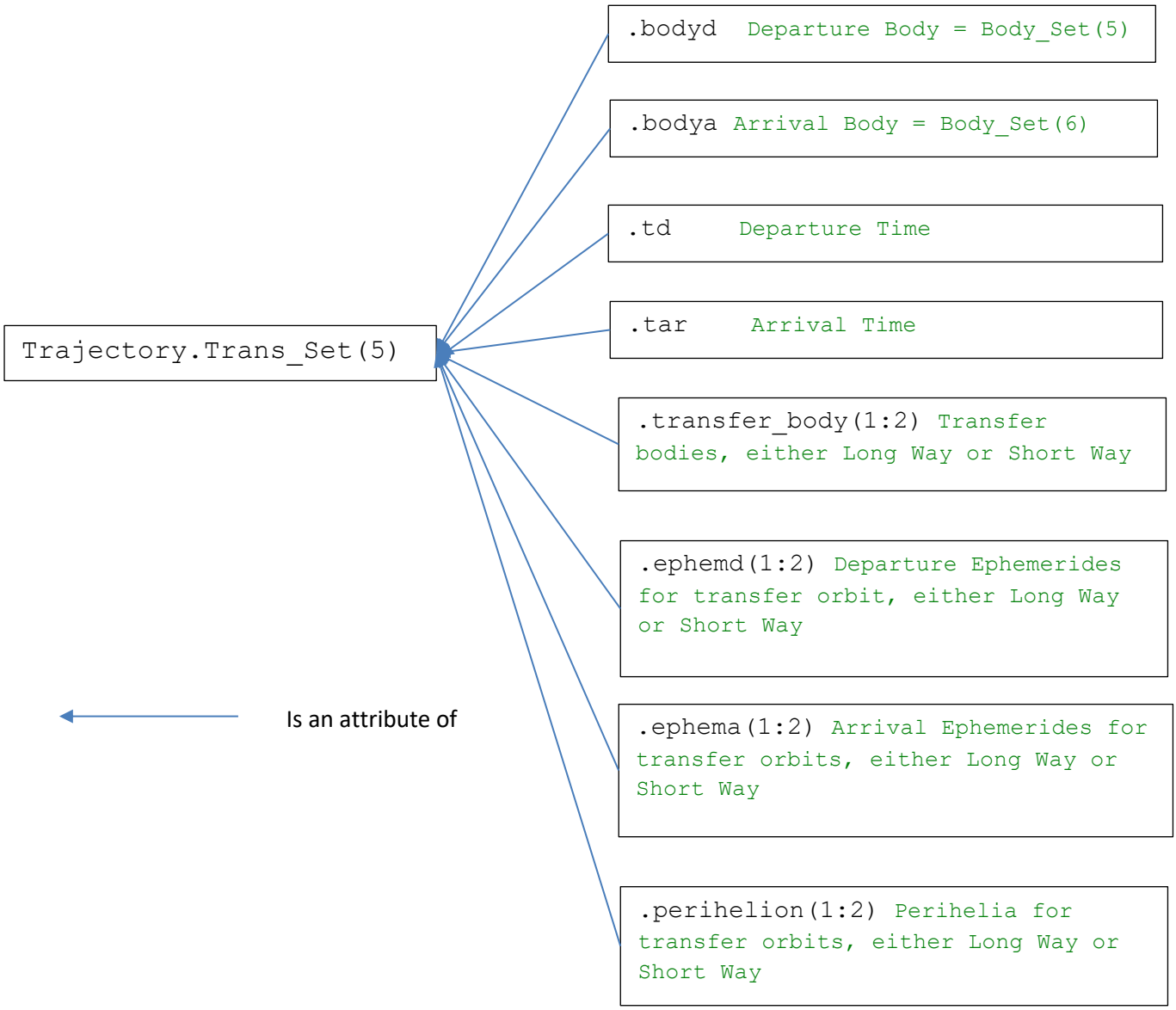


1.3.3) Outline of *Trajectory* Attribute *Trans_Set*

The Body subsequent to Mars (*Trajectory.Body_Set(5)*) along the interplanetary trajectory is Earth (*Trajectory.Body_Set(6)*). These two bodies are connected via a transfer trajectory denoted by *Solution.Trans_Set(5),* which is of type *Transfer_orbit.* The problem of connecting two points A & B, whose positions and times are known is the Lambert Problem and this can be solved by the Universal Variable theorem, which is elaborated in Reference (iii). This problem has two solutions, long way and short way.

More detail to *Solution.Trans_Set(5)* for JUICE is provided in Figure 1.3.3.1.

Figure 1.3.3.1



So the four most important components to *Trans_Set(5)* are the inputs:

```
Trans_Set(5).bodyd          ! MARS
Trans_Set(5).bodya          ! EARTH
Trans_Set(5).td             ! Departure time from MARS
Trans_Set(5).tar            ! Arrival time at EARTH
```

Having specified these inputs, there are two paths a transfer body might follow to start from Mars and arrive at Earth at the respective times. These can be represented as two possible incarnations of the transfer body, which are called *Trans_Set(5).transfer_body(1)* and *Trans_Set(5).transfer_body(2).* These transfer bodies have orbits and ephemerides in just the same way the encounter bodies defined by Body_Set do (refer Figure).

1.3.4) Various Velocity Increment Attributes of *Trajectory*

From the previous section we see that there are two transfer bodies (i.e. two possible spacecraft routes) from the 5th to 6th planets, i.e. from Mars to Earth. Also from Section 1.3.1, we know there are *Solution.Trajectory.Ntrans = 6* consecutive pairs. If we take the complete set of transfers along the JUICE trajectory, we find there is a total of $2^{Ntrans} = 2^6 = 64$ possible permutations of interplanetary trajectory. Let us designate the two possible ways, way 1 and way 2, this then allows us to define the attribute *perm* of *Trajectory*, as an array which contains all the possible permutations of Interplanetary Trajectory. Refer Table 1.3.4.1 for the definition of *perm* in the case of JUICE.

For each permutation *perm(1:64)* there is a corresponding *ΔV*, as follows:

```
Trajectory.DeltaV(1:64)
```

And the minimum of these is the output minimum *ΔV*, as follows:

```
Trajectory.BestDeltaV          ! = min(Trajectory.DeltaV(:))
```

Furthermore the best permutation which corresponds to this minimum *ΔV*, is as follows:

```
Trajectory.Best                ! = optimal permutation
```

Table 1.3.4.1

	perm(:,1)	perm(:,2)	perm(:,3)	perm(:,4)	perm(:,5)	perm(:,6)
	earth-earth	earth-venus	venus-earth	earth-mars	mars-earth	earth-jupiter
perm(1,:)	1	1	1	1	1	1
perm(2,:)	2	1	1	1	1	1
perm(3,:)	1	2	1	1	1	1
perm(4,:)	2	2	1	1	1	1
perm(5,:)	1	1	2	1	1	1
perm(6,:)	2	1	2	1	1	1
perm(7,:)	1	2	2	1	1	1
perm(8,:)	2	2	2	1	1	1
perm(9,:)	1	1	1	2	1	1
perm(10,:)	2	1	1	2	1	1
perm(11,:)	1	2	1	2	1	1
perm(12,:)	2	2	1	2	1	1
perm(13,:)	1	1	2	2	1	1
perm(14,:)	2	1	2	2	1	1
perm(15,:)	1	2	2	2	1	1
perm(16,:)	2	2	2	2	1	1
perm(17,:)	1	1	1	1	2	1
perm(18,:)	2	1	1	1	2	1
perm(19,:)	1	2	1	1	2	1
perm(20,:)	2	2	1	1	2	1
perm(21,:)	1	1	2	1	2	1
perm(22,:)	2	1	2	1	2	1
perm(23,:)	1	2	2	1	2	1
perm(24,:)	2	2	2	1	2	1
perm(25,:)	1	1	1	2	2	1
perm(26,:)	2	1	1	2	2	1
perm(27,:)	1	2	1	2	2	1
perm(28,:)	2	2	1	2	2	1
perm(29,:)	1	1	2	2	2	1
perm(30,:)	2	1	2	2	2	1
perm(31,:)	1	2	2	2	2	1
perm(32,:)	2	2	2	2	2	1
perm(33,:)	1	1	1	1	1	2
perm(34,:)	2	1	1	1	1	2
perm(35,:)	1	2	1	1	1	2
perm(36,:)	2	2	1	1	1	2
perm(37,:)	1	1	2	1	1	2
perm(38,:)	2	1	2	1	1	2
perm(39,:)	1	2	2	1	1	2
perm(40,:)	2	2	2	1	1	2
perm(41,:)	1	1	1	2	1	2
perm(42,:)	2	1	1	2	1	2
perm(43,:)	1	2	1	2	1	2
perm(44,:)	2	2	1	2	1	2
perm(45,:)	1	1	2	2	1	2
perm(46,:)	2	1	2	2	1	2
perm(47,:)	1	2	2	2	1	2
perm(48,:)	2	2	2	2	1	2
perm(49,:)	1	1	1	1	2	2
perm(50,:)	2	1	1	1	2	2
perm(51,:)	1	2	1	1	2	2
perm(52,:)	2	2	1	1	2	2
perm(53,:)	1	1	2	1	2	2
perm(54,:)	2	1	2	1	2	2
perm(55,:)	1	2	2	1	2	2
perm(56,:)	2	2	2	1	2	2
perm(57,:)	1	1	1	2	2	2
perm(58,:)	2	1	1	2	2	2
perm(59,:)	1	2	1	2	2	2
perm(60,:)	2	2	1	2	2	2
perm(61,:)	1	1	2	2	2	2
perm(62,:)	2	1	2	2	2	2
perm(63,:)	1	2	2	2	2	2
perm(64,:)	2	2	2	2	2	2

1.3.5) Outline of *Trajectory* Attribute *Hyperbola*

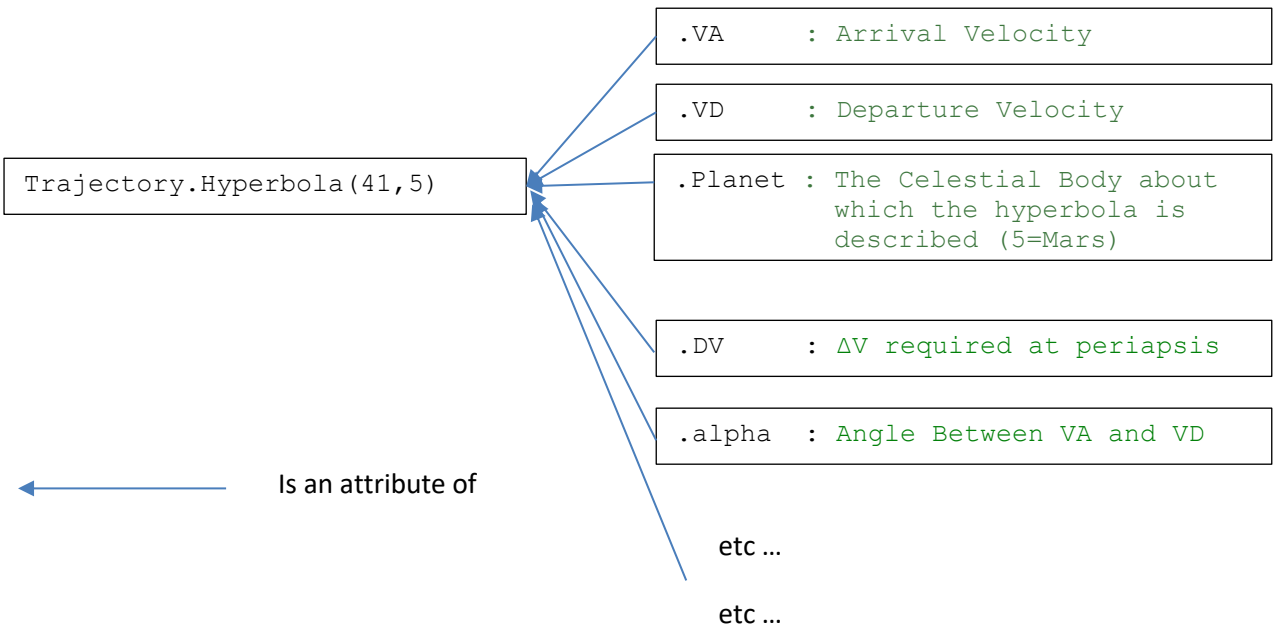
If *Trajectory.Nbody > 2* (or alternatively *Trajectory.Ntrans > 1*) then the *Hyperbola* attribute is defined and used, otherwise this attribute is defined but unused. *Hyperbola* is an array of objects of class *Connecting_Hyperbola*. These objects express the encounter dynamics of the spacecraft w.r.t. each non-terminal SSO along the interplanetary trajectory, in other words the incoming and outgoing hyperbolae for each gravitational assist. Since neither the initial or final SSO will be a gravitational assist (by definition), *Hyperbola* is only used for interior SSO visits.

Hyperbola is a 2D array of objects, where the first (row) dimension indicates the number of permutation of *perm* under consideration, and the second (column) dimension represents the number of the SSO in the sequence under consideration. Thus we have for JUICE the following:

```
Hyperbola(i,j) !  i=1:64, corresponding to the permutation perm under
               !  consideration
               !  j=2:6,  corresponding to GA at each interior planet,i.e.
               !  j=2, Earth,
               !  j=3, Venus,
               !  j=4, Earth,
               !  j=5, Mars and
               !  j=6, Earth
```

The main attributes of *Trajectory.Hyperbola* are shown in Figure 1.3.5.1., corresponding to the 41st permutation of *Trajectory.perm*, which happens to be the value of *Trajectory.Best*, i.e. for the optimal permutation, *Trajectory.Best = 41*.

Figure 1.3.5.1



1.3.6) Some More Attributes of *Solution*

In addition to all the attributes elaborated in the previous sections 1.3.1 to 1.3.5, there are a few others which are worth addressing, these are shown below:

```
TotaldV;           % Total DeltaV of Mission
FlybyRendez=0;     % Flyby or Rendezvous Flag =0 (FLYBY) = 1 (RENDEZVOUS)
wayflag=1;         % Default to Prograde Only
home_periapsis;    % Periapsis radial distance from Home's Centre
target_periapsis; % Periapsis radial distance from Target's Centre
```

Solution.TotaldV is simply set equal to the value of *Solution.Trajectory.BestDeltaV* (see section 1.3.4).

Solution.FlybyRendez sets whether there is a burn at the target planet (for JUICE that is Jupiter) to stay with the target (*Solution.FlybyRendez=1*) or whether the spacecraft continues on its journey WITHOUT a ΔV to match velocities with the target (*Solution.FlybyRendez=0*).

Solution.wayflag=1 is the default and specifies that the transfers between each pair of consecutive planets will be prograde.

If *Solution.home_periapsis==0* then the ΔV at the home planet is equivalent to the required hyperbolic excess speed at the home planet, otherwise this ΔV is the velocity increment needed in excess of that needed for a circular orbit placed at an altitude of *Solution.home_periapsis* above the home planet's surface.

Solution.target_periapsis is only relevant where 'Rendezvous with Destination' is selected (i.e. *Solution.FlybyRendez=1*). When the latter is selected then if *Solution.target_periapsis==0*, the ΔV at the target planet is simply the hyberbolic arrival speed with respect to the target planet (for JUICE this target planet is Jupiter). Otherwise it is the change in velocity required at the given altitude with respect to the target planet, this altitude being specified by *Solution.target_periapsis*, in order that the resulting hyperbolic excess of ecape at the target is zero.

2) Dynamic Overview

This Section describes how a typical run of OITS might proceed and the corresponding tasks which are performed by OITS. A typical OITS flow is obviously governed by user inputs and two typical examples are shown in Figure 2.1. This is summarized below.

A When the command *run_OITS* is entered, a *global* object of class *Project* is created, called *This*.

B The control is then taken over by *OITS.m* which displays the Main Menu GUI.

B1 Assuming the user selects 'NEW MISSION' from the choice available in **B**, control is maintained by OITS.m and the user selects the sequence of SSOs to be encountered for the mission in question

C Having set the general mission parameters in **B1**, the user then selects 'SET OPTIMIZER DETAILS' upon which control is transferred to *Set_Mission.m* and its GUI

D Having quit the *Set_Mission.m* GUI, control returns to *OITS.m* where the user selects 'OPTIMIZE MISSION'

E or F On completion of the optimization in **D**, control returns to *OITS.m* where the user selects 'ANIMATE RESULTS', alternatively he may select 'VIEW RESULTS'

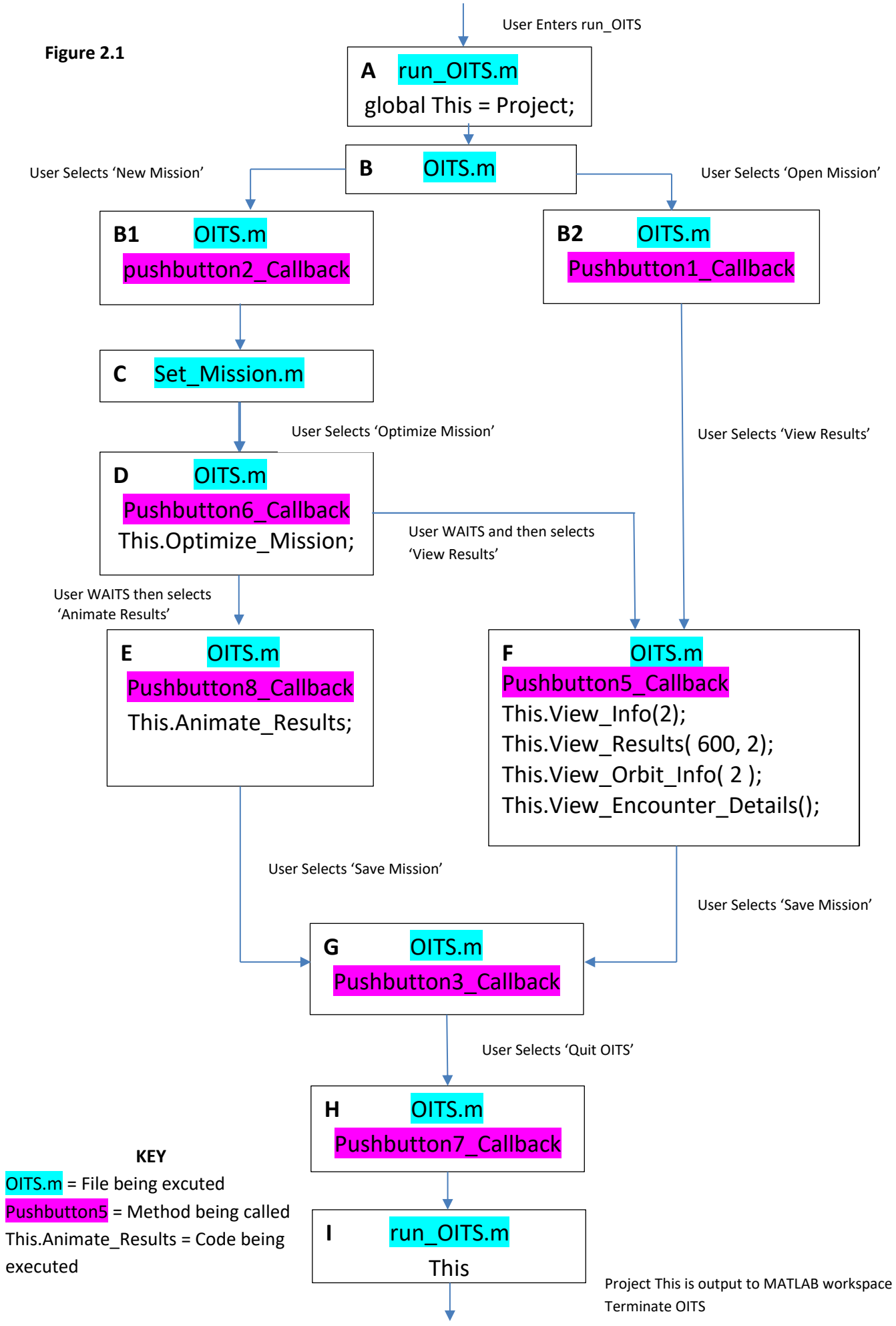
G Having analysed results from either **E** or **F**, control returns to *OITS.m* where the user selects 'SAVE MISSION'

H The user selects 'QUIT OITS'

I The Project *This* is then output to the MATLAB workspace to be analysed in detail by the user.

More detailed functionality is explained in the following sections.

Figure 2.1


User Enters run_OITS
A run_OITS.m
global This = Project;
B OITS.m
User Selects 'New Mission'
User Selects 'Open Mission'
B1 OITS.m
pushbutton2_Callback
B2 OITS.m
Pushbutton1_Callback
C Set_Mission.m
User Selects 'Optimize Mission'
User Selects 'View Results'
D OITS.m
Pushbutton6_Callback
This.Optimize_Mission;
User WAITS and then selects 'View Results'
User WAITS then selects 'Animate Results'
E OITS.m
Pushbutton8_Callback
This.Animate_Results;
F OITS.m
Pushbutton5_Callback
This.View_Info(2);
This.View_Results(600, 2);
This.View_Orbit_Info(2);
This.View_Encounter_Details();
User Selects 'Save Mission'
User Selects 'Save Mission'
G OITS.m
Pushbutton3_Callback
User Selects 'Quit OITS'
H OITS.m
Pushbutton7_Callback
I run_OITS.m
This
Project This is output to MATLAB workspace
Terminate OITS
KEY
OITS.m = File being excuted
Pushbutton5 = Method being called
This.Animate_Results = Code being executed

2.1) run_OITS.m

First of all *run_OITS.m* is executed by typing *run_OITS* in the MATLAB command window.

run_OITS.m initialises OITS and configures the global object *This* (of class *Project*) ready for user specification and application.

run_OITS.m undertakes the following tasks in the following order:

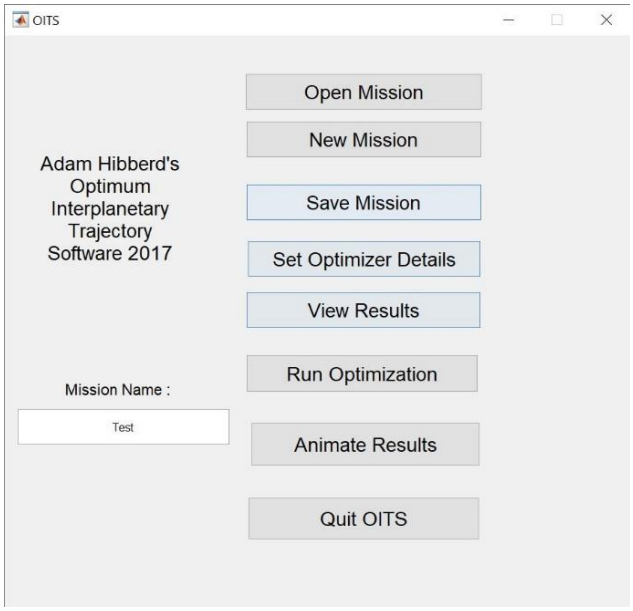
1) Initialise directory paths needed for OITS
2) Declare *This* as a global object of class *Project*
3) Perform some basic initialisations of *This*
4) Initialise the directory paths for NLOPT (see Section 1.2.1) Required Directory Structure and Files
5) MEX NLOPT
6) Initialize SPICE
7) Update list of available SPICE bodies using their relevant **.bsp* file (see Section1.2.2) NASA SPICE toolkit & Binary SPICE Kernel Files
8) Include Intermediate Points and Fixed Points in the list of available bodies
9) Merge data for each planet (Gravitational Mass, Radius, etc) with their respective SPICE bodies
10) Start the Main Menu GUI (`figure(OITS);`) and hand control over to the user
11) The user having finished with *This*, update the MATLAB workspace with this object.

2.2) OITS.m

OITS.m controls the GUI of OITS (the *OITS.fig* file provides the GUI format and arrangement). For detailed operation of OITS, refer to Reference (ii).

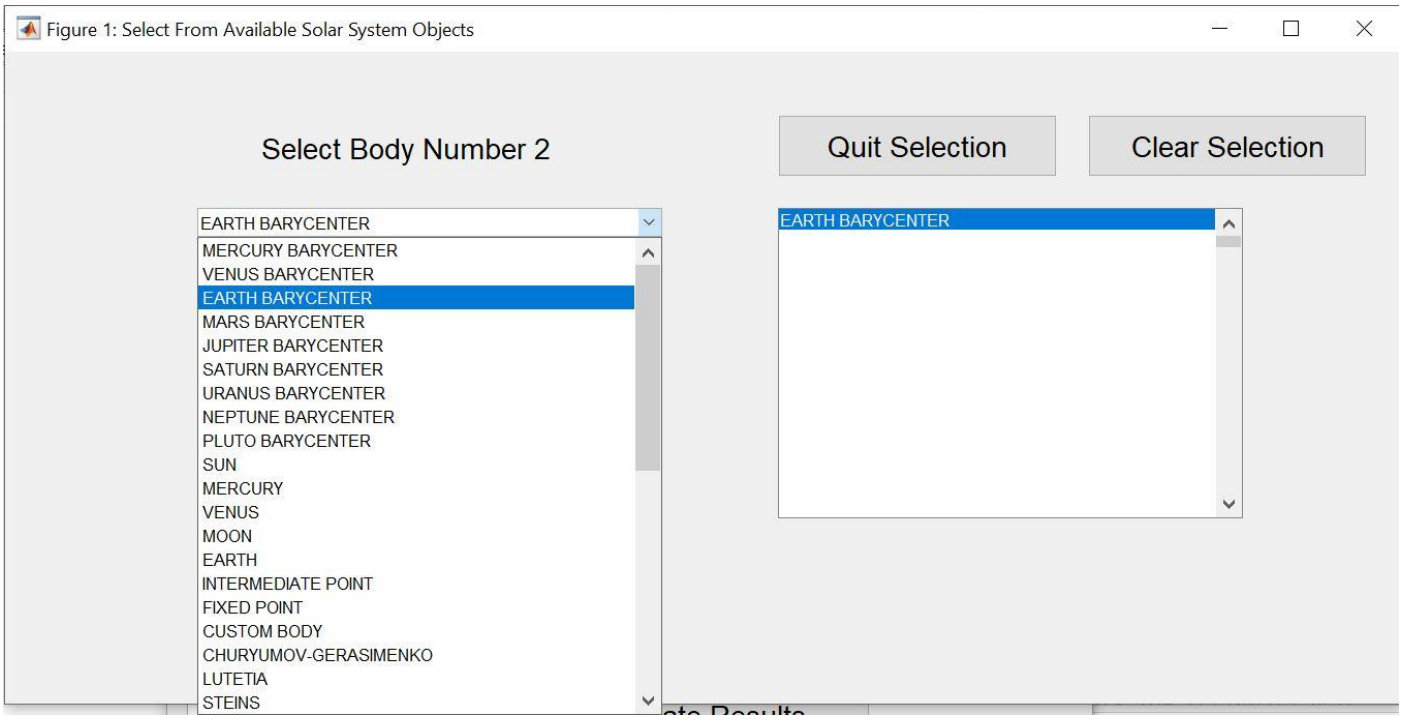
The main menu is shown in Figure 2.2.1.

Figure 2.2.1



OITS.m organises the GUI when 'NEW MISSION' is selected which is shown in Figure 2.2.2.

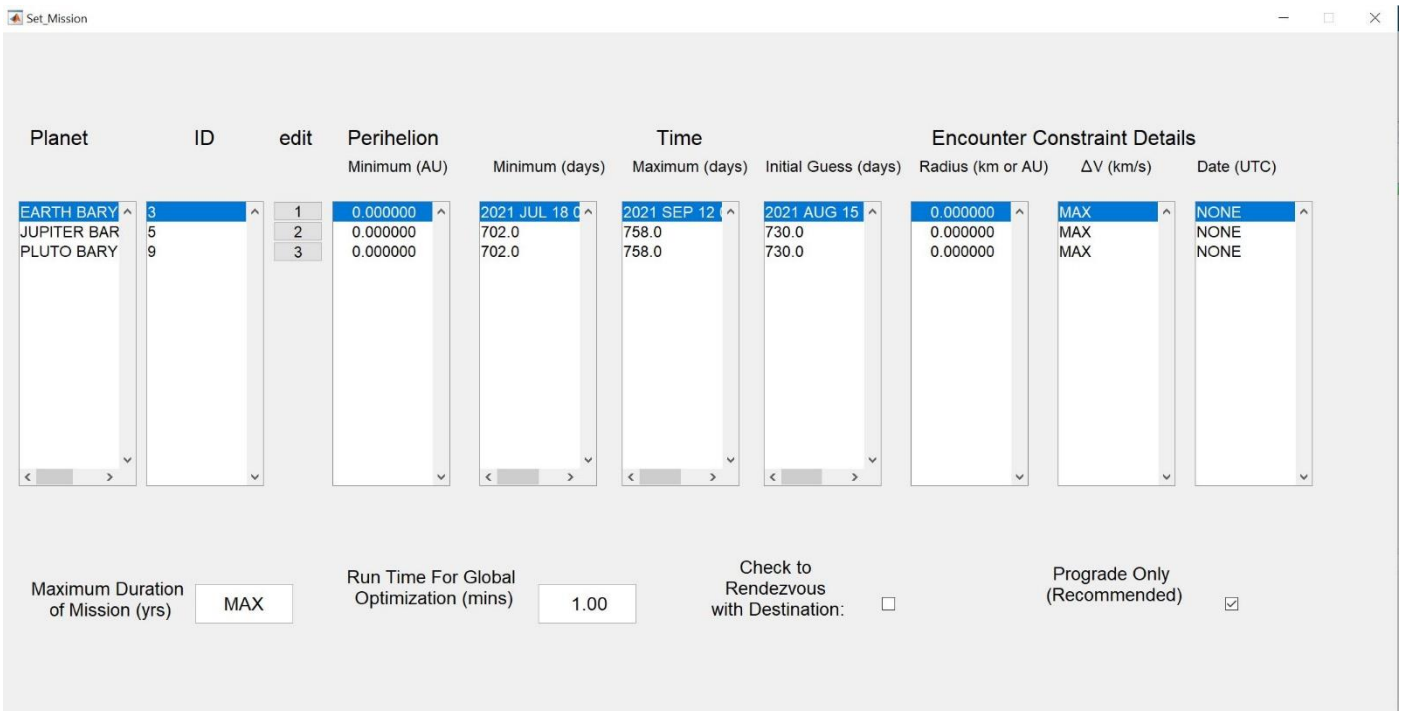
Figure 2.2.2



2.3) Set_Mission.m

Set_Mission.m GUI is called from *OITS.m* on selection of SET OPTIMZER DETAILS. The corresponding *Set_Mission.fig* file specifies the visual arrangement and format of the GUI which is shown in Figure 2.3.1.

Figure 2.3.1



2.4) Optimize_Mission

This Section goes into the detail as to how a mission is optimized, precisely what are the salient classes/objects and methods?

2.4.1) Overview

There are two fundamental requirements for *Optimize_Mission* to work:

- The parameters of the *Project* object currently under consideration must have been appropriately defined and initialised (refer Section (2.2) OITS.m, Section (2.3) Set_Mission.m and also document (ii)). In the case of OITS, the *Project* object is *This*, which is global in scope.
- In particular the *Current_Mission* attribute of the *Project* object under consideration must also have been appropriately defined (refer Section (2.2) OITS.m, Section (2.3) Set_Mission.m and also document (ii)).

2.4.2) Description of Operation

Refer Figure 2.4.2.1 for more detailed description of how *Optimize_Mission* works.

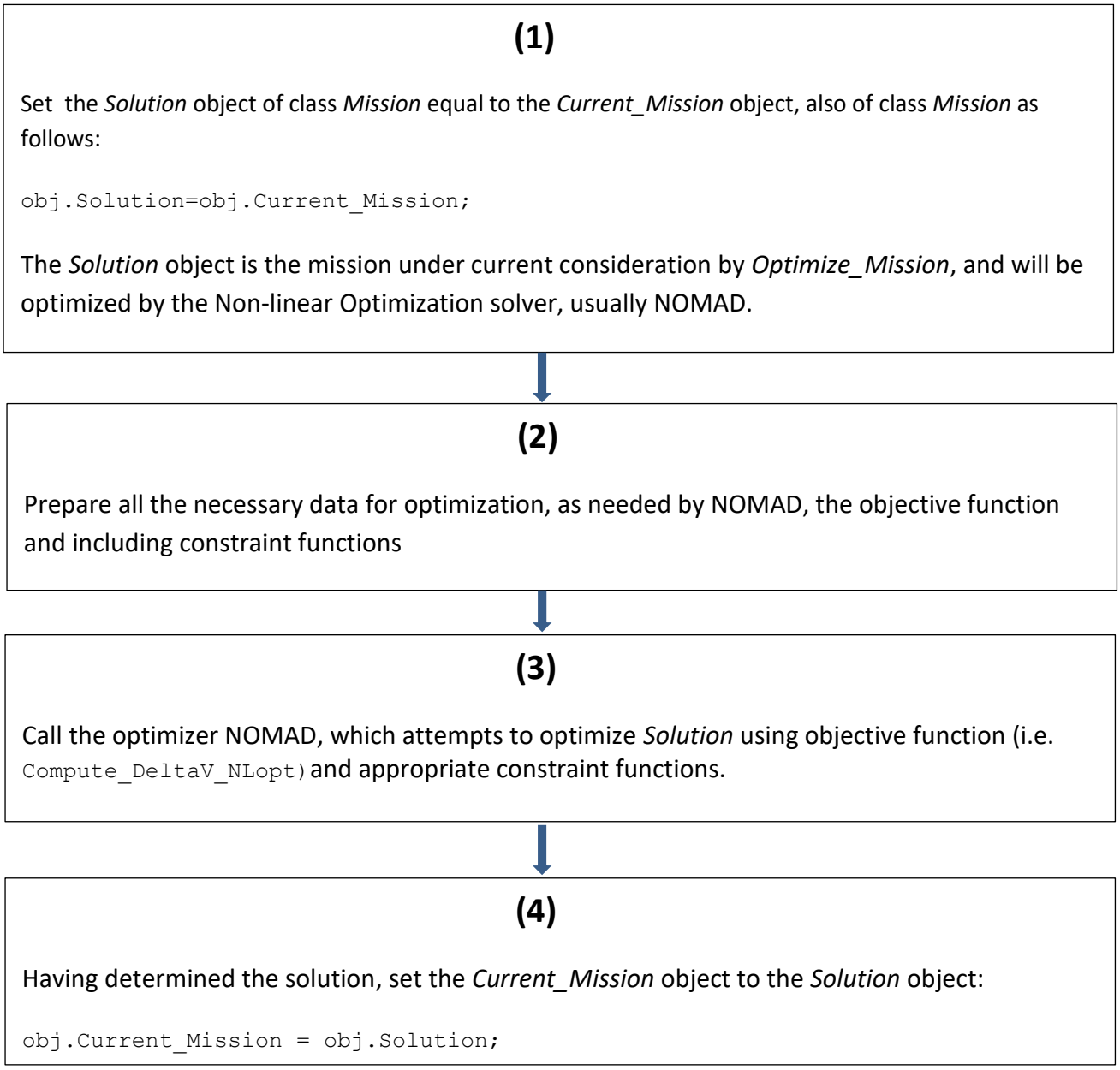
Let us delve into the process by which **(2)** is achieved in Figure 2.4.2.1.

Firstly a quick note as to the functions which are called as a part of the optimization process, i.e. they are called either directly by the NOMAD solver, or by a function which is called by the NOMAD solver (in the case of *Update_Traj*). They are listed in Table 2.4.2.1 below and are sub-routines of *Optimize_Mission*:

Table 2.4.2.1

	Function	Purpose
A	`function [DeltaV, gradient ] = Compute_DeltaV_NLopt(tin)`	Objective Function (ΔV)
B	`function [DeltaV,gradient] = Update_Traj(tin)`	Updates Trajectory
C	`function [ con, gradient ] = Per_NLopt(tin,run_mode)`	Min/Max Periapsis Altitude for each encounter
D	`function [ con, gradient ] = dV_NLopt(tin,run_mode)`	Min/Max ΔV for each encounter where specified
E	`function [ con, gradient ] = TI_NLopt(tin,run_mode)`	Min/Max Absolute Time of periapsis for each encounter where specified
F	`function [ con, gradient ] = Perhel(tin,run_mode)`	Min Perihelia for each Interplanetary Transfer where specified
G	`function [ cond, gradient ] = Overall_Duration(tin)`	Maximum Overall Mission Duration, where specified

Figure 2.4.2.1



We have seen that in **(1)**, the value of *Solution* is set equal to *Current_Mission*. A notable attribute of *Solution* is the required *Trajectory* to be optimized, which is of type *Nbody_Trajectory_With_Encounters*:

`obj.Solution.Trajectory`

Furthermore Trajectory has the following notable attributes:

`obj.Solution.Trajectory.Nbody`	The number of bodies visited along the Trajectory
`obj.Solution.Trajectory.Body_Set(i)`	The set of bodies (of type Body) to be visited along the Trajectory, i=1:Nbody

The *Body_Set(i)* array can be a celestial body for which binary SPICE kernel data has been provided, it can be a CUSTOM BODY whose orbit has been fully specified by the user, or alternatively it can be an Intermediate Point (IP). The latter, IP, is determined by the attribute *Fixed_Point* of *Body* as follows:

```
obj.Solution.Trajectory.Body_Set(i).Fixed_Point == 1
```

Optimize_Mission starts off by counting the number of bodies which are IP's along the trajectory.

Having performed this task, it then initialises all the bodies along the trajectory where the optimizer will find the optimal point along the body's orbit rather than necessarily intercept the body at the correct point in that orbit. This setting is triggered by the word 'ORBIT' or lowercase 'orbit' included in the name of the body as follows:

```
contains(obj.Solution.Trajectory.Body_Set(i1).name,"ORBIT",'IgnoreCase',true)
```

Having determined which bodies contain this word, this is flagged to the subroutine `Update_Traj`, i.e. B in Table 2.4.2.1 optimizer (B will be called while the optimization process is underway) as follows:

```
obj.Solution.Trajectory.Body_Set(i1).Fixed_Point=-1;
```

Next the constraint functions for the NOMAD optimizer, along with associated data, must be initialised where the user has specified them.

2.4.3) Periapsis Constraints

Firstly the periapsis constraints are specified, these being pertinent only when *Nbody > 2* (hence Gravitational Assists - GAs – are only used when there is at least one planetary encounter which is neither at the beginning nor the end). The number of non-terminal bodies is stored in *Nconstraints*, an attribute of *Project* as follows:

```
obj.Nconstraints = obj.Current_Mission.Trajectory.Nbody - 2 ;
```

For each non-terminal body, two periapsis constraints are necessary:

- firstly the periapsis must be greater than some value previously specified by the user, and
- secondly the periapsis must be less than the Laplace Sphere of Influence for the body in question (ref. (iii)). The latter constraint is imposed to help prevent undershoot of the minimum periapsis distance, and is present by default - it is not modifiable by the user.

Both of the above periapsis constraints are handled by constraint function `Per_NLopt(tin,run_mode)` (i.e. **C** in Table 2.4.2.1). To understand how `Per_NLopt(tin,run_mode)` works let us assume we have *Nbody=7*, like for example is the case of the ESA JUpiter Icy Moon Explorer mission, JUICE, to be launched in 2022. The sequence of bodies encountered is Earth, Earth, Venus, Earth, Mars, Earth, Jupiter. See Table 2.4.3.1 which summarises the inputs to `Per_NLopt`.

Table 2.4.3.1

i >	1	2	3	4	5	6	7
Body_Set(i)>	**Earth**	**Earth**	**Venus**	**Earth**	**Mars**	**Earth**	**Jupiter**
Minimum Periapsis >	N/A	Per_NLopt(tin,2)	Per_NLopt(tin,3)	Per_NLopt(tin,4)	Per_NLopt(tin,5)	Per_NLopt(tin,6)	N/A
Maximum Periapsis >	N/A	Per_NLopt(tin,7)	Per_NLopt(tin,8)	Per_NLopt(tin,9)	Per_NLopt(tin,10)	Per_NLopt(tin,11)	N/A

2.4.4) Perihelion Constraints

There are two types of Perihelion Constraints implemented in OITS, depending on user inputs (refer (ii)). The two options are:

1. the minimum distance from the sun along a specified *Transfer_orbit* (*Trans_Set(i).perihelion*) must be greater than a minimum, as specified by the user
2. the orbital parameter associated with a particular *Transfer_orbit* (where the orbital parameters are defined in *Trans_Set(i).transfer_body(1:2).orbit*) must be greater than a minimum, as specified by the user.

An integer flag attribute of Project *Perihelia_flag* signifies which of the situations 1 or 2 above is applicable. Thus if the bit n of *Perihelia_flag* is set, then that means the perihelion constraint 2 above is in operation between the transfer from body n-1 to body n, otherwise it is constraint 1 above.

The *Perihelia* attribute of Project is an array of minimum Perihelia where if the transfer from body n-1 to body n has a constraint applied then *Perihelia(n)* is the value of that constraint.

For the purposes of illustration, let us suppose that for JUICE the mission, transfer 2-3 (Earth to Venus) and transfer 5-6 (Mars to Earth), both have perihelia constraints but the latter transfer has been specified as option 2 above. Then Table 2.4.4.1 shows the values of *Perihelia* and the bits of *Perihelia_flag*.

Table 2.4.4.1

i >	1	2	3	4	5	6	7
Body_Set(i)>	**Earth**	**Earth**	**Venus**	**Earth**	**Mars**	**Earth**	**Jupiter**
Transfer Number>	**1**	**2**	**3**	**4**	**5**	**6**	
Minimum Periapsis >	N/A	-	Perhel(tin,3)	-	-	Perhel(tin,6)	-
Perihlelia_flag>	0	0	0	0	0	1	0

2.4.5) Encounter Time Constraints

As explained in Section 1.3.1, it is the purpose of the optimizer to attempt to optimize the following array:

```
Solution.Mission_Times(i)   i=1:Solution.Trajectory.Nbody
```

Just to reiterate here, the example mission is JUICE and for this mission, *Mission_Times* is an array of encounter times for each of the Bodies on the left-hand side of Figure 1.3.1.1 in turn. In particular, M*ission_Times(1)* is the absolute time, TBD (Barycentric Dynamical Time) in seconds. Subsequent times are Times-of-Flight (TOF).

Although the purpose of the optimizer is to find optimal values of *Mission_Times*, there may be some requirement for an encounter to be no later or alternatively no earlier than some user-specified absolute encounter time.

```
Solution.Absolute_Times(i)  i=1:Solution.Trajectory.Nbody
```

Furthermore *Absolute_Times* is an array of absolute encounter times for each of the bodies on the left-hand side of Figure 1.3.1.1 in turn.

Con_TI(i) (attribute of *Project*) is an array of absolute encounter times where i represents the number of the body to which the constraint applies. By default, no such constraint is set for any value of i, in which case *Con_TI(i) = -1e50* for all i. If however *Con_TI(i) > -1e50*, then that signifies the user has specified such a contraint for encounter i. Note furthermore that by default, this constraint is a maximum absolute time constraint. If however the flag *Min_TI_flag* (another attribute of Project) has bit (i-1) set, then the encounter time constraint is a minimum absolute time.

2.4.6) ΔV Constraints

A ΔV constraint at a particular body i can either be a maximum (in which case the *Project* attribute *Max_dV(i)* is greater than zero in which case *Max_dV(i)* represents the maximum value of ΔV for body i) or a minimum (in which case the *Project* attribute *Max_dV(i)* is less than zero and the absolute value of *Max_dV(i)* represents the minimum value of ΔV for body i). If no constraint is present then *Max_dV(i) ≥ 1e50.*