Debris Removal Tool

Release Beta

Gaetan PIERRE, Come OOSTERHOF, Tom SEMBLANET, Myrtille M

Feb 09, 2022

CONTENTS:

Debris Removal Tool (DRT) is a sample of python codes aiming to plan debris removal missions (consisting in multiple space rendezvous). For a given set of debris, DRT plans several missions, each one including 4 to 5 debris (by default).

It is focused on two major axes which are :

- the regroupement of the debris in order to plan optimal missions
- the plannification of the multiple rendezvous missions

Note: This project is under development.

CHAPTER

ONE

REGROUPEMENT

1.1 dV_computations

1.1.1 Time evaluation for J2 perturbation transfer

Created on Wed Jan 26 18:09:33 2022

@author: g.pierre

`regroupement.dV_computations.compute_dt_alignment.`**`compute_dt`**(*a1*, *a2*, *i1*, *i2*, *RAAN1*, *RAAN2*, *print_result=False*)

Function computing the delta_t [s] required to modify the RAAN of the orbit from RAAN1 to RAAN2

Arguments [] a1 (float) : Initial SMA [km]

a2 (float) : Final SMA [km]

i1 (float) : Initial inclination [rad]

i2 (float) : Final inclination [rad]

RAAN1 (float) : Initial Right ascension of the ascending node [rad]

RAAN2 (float) : Final Right ascension of the ascending node [rad]

Returns [] dt_days (float) : Required delta_t [days]

`regroupement.dV_computations.compute_dt_alignment.`**`compute_dt_matrix`**(*debris_data*)

Function computing the delta_t matrix of all possible transfer from a debris to another

Arguments [] debris_data (dataframe) : Orbital parameters of the debris considered

Returns [] dt_matrix (array) : Matrix whose (i,j) indice represents the delta_t (in seconds) required to modify the RAAN of the orbit from the RAAN of the debris i to the RAAN of the debris j

1.1.2 Computation of the delta-v associated to a maneuver

This module computes the delta-Vs necessary to change orbital parameters :

- SMA (semi-major axis)
- ECC (eccentricity)
- INC (inclination)
- AOP (argument of perigee)
- RAAN (right ascension of the ascending node)

- TA (true anomaly)

`regroupement.dV_computations.maneuvers_dV.`**`AOP_dV`**`(`*w1*, *w2*, *RAAN*, *a*, *e*, *i*, *m*`)`

Computes the delta-V [km/s] required to modify the AOP of the orbit from w1 to w2

Arguments [] w1 (float) : Initial AOP [rad]

w2 (float) : Final AOP [rad]

RAAN (float) : Right ascension of the ascending node [rad]

a (float) : Semi-major axis [km]

e (float) : Eccentricity [-]

i (float) : Inclination [rad]

m (float) : Body's mass [kg]

Returns [] dV (float) : Required delta-V [km/s]

`regroupement.dV_computations.maneuvers_dV.`**`INC_dV`**`(`*i1*, *i2*, *V1*`)`

Computes the delta-V [km/s] required to modify the INC of the orbit from i1 to i2

Arguments [] a1 (float) : Initial INC [rad]

a2 (float) : Final INC [rad]

V1 (float) : Velocity on the orbit (supposed to be circular)

Returns [] dV (float) : Required delta-V [km/s]

`regroupement.dV_computations.maneuvers_dV.`**`SMA_dV`**`(`*a1*, *a2*`)`

Computes the delta-V [km/s] required to modify the SMA of the orbit from ai to af

Arguments [] a1 (float) : Initial SMA [km]

a2 (float) : Final SMA [km]

Returns [] dV (float) : Required delta-V [km/s]

1.1.3 Computation of the dV matrix used for the stochastic optimization

This module computes the delta-Vs necessary to change orbital parameters :

- SMA (semi-major axis)
- ECC (eccentricity)
- INC (inclination)
- AOP (argument of perigee)
- RAAN (right ascension of the ascending node)
- TA (true anomaly)

`regroupement.dV_computations.maneuvers_dV.`**`AOP_dV`**`(`*w1*, *w2*, *RAAN*, *a*, *e*, *i*, *m*`)`

Computes the delta-V [km/s] required to modify the AOP of the orbit from w1 to w2

Arguments [] w1 (float) : Initial AOP [rad]

w2 (float) : Final AOP [rad]

RAAN (float) : Right ascension of the ascending node [rad]

a (float) : Semi-major axis [km]

e (float) : Eccentricity [-]

i (float) : Inclination [rad]

m (float) : Body's mass [kg]

Returns [] dV (float) : Required delta-V [km/s]

`regroupement.dV_computations.maneuvers_dV.`**`INC_dV`**(*i1*, *i2*, *V1*)

Computes the delta-V [km/s] required to modify the INC of the orbit from i1 to i2

Arguments [] a1 (float) : Initial INC [rad]

a2 (float) : Final INC [rad]

V1 (float) : Velocity on the orbit (supposed to be circular)

Returns [] dV (float) : Required delta-V [km/s]

`regroupement.dV_computations.maneuvers_dV.`**`SMA_dV`**(*a1*, *a2*)

Computes the delta-V [km/s] required to modify the SMA of the orbit from ai to af

Arguments [] a1 (float) : Initial SMA [km]

a2 (float) : Final SMA [km]

Returns [] dV (float) : Required delta-V [km/s]

1.2 Optimizer

1.2.1 Initialization of the first state

Created on 08/12/2021

@author: Yvan GARY

`regroupement.optimizer.Init_alea_G.`**`Init_alea_G`**(*nb_debris*, *s_min*, *s_max*, *DV*, *DT*)

Function used to initiate the optimization

Arguments: nb_debris (int): Number of debris in the given catalogue

s_min (int) : Minimum number of debris contained in a group

s_max (int) : Maximum number of debris contained in a group

DV (Matrix): Matrix containing the delta_v associated to each maneuver

DT (Matrix): Matrix containing the elapsed time associated to each "J2 perturbation duration" between two debris

Returns: G (matrix): First state generated randomly to begin Optimization

E (float): Energy associated to the state G

1.2.2 Computation of the energy associated to a state

Created on 09/12/2021

@author: Yvan GARY

`regroupement.optimizer.energy_computation.`**`energy_computation`**(*G*, *DV*, *DT*, *detail=False*)

Function used to compute the energy associated to a state

Arguments: G (Matrix): Actual state for which we compute the energy

DV (Matrix): Matrix containing the delta_v associated to each maneuver

DT (Matrix): Matrix containing the elapsed time associated to each "J2 perturbation duration" between two debris

detail (bool) : False by default - If True, gives the detail of the energy for each group

Returns: E (float): Energy associated to the state G

E_transfers (array) - optionnal : Energy associated to each individual group

E_transfers_dV (array) - optionnal : delta v associated to each individual group

E_transfers_dt (array) - optionnal : elapsed time due to J2 perturbation associated to each individual group

`regroupement.optimizer.energy_computation.`**`energy_computation_DT`**(*G*, *DT*, *detail=False*)

Function used to compute the delta t (J2) associated to a state

Arguments: G (Matrix): Actual state for which we compute the energy

DV (Matrix): Matrix containing the delta_v associated to each maneuver

detail (bool) : False by default - If True, gives the detail of the delta v for each group

Returns: dV (float): Global delta v associated to the state G

dV_transfers (array) - optionnal : Global delta v associated to each individual group

`regroupement.optimizer.energy_computation.`**`energy_computation_DV`**(*G*, *DV*, *detail=False*)

Function used to compute the delta v associated to a state

Arguments: G (Matrix): Actual state for which we compute the energy

DV (Matrix): Matrix containing the delta_v associated to each maneuver

detail (bool) : False by default - If True, gives the detail of the delta v for each group

Returns: dV (float): Global delta v associated to the state G

dV_transfers (array) - optionnal : Global delta v associated to each individual group

1.2.3 Implementation of the dynamic of Metropolis

Created on 08/12/2021

@author: Yvan GARY

`regroupement.optimizer.Metropolis.`**`Metropolis`**(*G_in*, *E_in*, *s_min*, *s_max*, *DV*, *DT*, *T*)

Function computing the dynamic of Metropolis. A neighbour of a state G is defined as a switch of two debris between two groups selected randomly. Then it is kept or abandonned according to the Metropolis dynamic.

Arguments: G_in (Matrix): Current state i.e. current regroupments of debris

E_in (float): Energy associated to the current state G_in

s_min (int) : Minimum number of debris contained in a group

s_max (int) : Maximum number of debris contained in a group

DV (Matrix): Matrix containing the delta_v associated to each maneuver

DT (Matrix): Matrix containing the elapsed time associated to each "J2 perturbation duration" between two debris

T (float) : Temperature related to the dynamic of Metropolis

Returns: G_out (Matrix): Output state of the dynamic of Metropolis

E_out (float): Energy associated to the new state G_out

`regroupement.optimizer.Metropolis.`**`select_random_debris`**(*G*, *grps*, *card_grps*)

Function selecting randomly one debris in each group in grps (used to compute neighbours)

Arguments: G (Matrix) : Current state i.e. current regroupments of debris

grps (1d-array) : Array containing the groups (of same size) indices

card_grps (int) : Number of debris contained in each group (the same for every group), typically s_max

Returns: selected_debris (1d-array) : Array containing the indices of the selected debris in each group (same order as groups)

`regroupement.optimizer.Metropolis.`**`split_and_fill`**(*G*, *grps*, *grp_idx*)

Function selecting randomly a group to split and groups to be filled (used to compute neighbours)

Arguments: G (Matrix) : Current state i.e. current regroupements of debris

grps (1d-array) : Array containing the groups (of same size) indices

grp_idx (1d_array) : indices of the group that can be filled (with cardinal < s_max)

Returns: G (Matrix) : Current state i.e. current regroupements of debris after distribution

1.2.4 Simulated Annealing

Created on 13/12/2021

@author: Yvan GARY

`regroupement.optimizer.Recuit.`**`Recuit`**(*nb_debris*, *s_min*, *s_max*, *DV*, *DT*, *Ti*, *Tf*, *alpha*, *n_classes*, *t_iter*, *n_iter*)

Function computing the simulated annealing, with the corresponding dynamic of Metropolis. It corresponds to the succession of Markov chains computed with decreasing temperatures. At the end we obtain a state G_out that minimizes the energy we defined, that is to say the sum of the delta_v associated to each groups. G_out contains the final groups that reach this minimal "global delta_v".

Arguments: nb_debris (int) : Nulber of debris in the given catalogue

s_min (int) : Minimum number of debris contained in a group

s_max (int) : Maximum number of debris contained in a group

DV (Matrix) : Matrix containing the delta_v associated to each maneuver

DT (Matrix): Matrix containing the elapsed time associated to each "J2 perturbation duration" between two debris

Ti (float) : Initial temperature related to the dynamic of Metropolis

Tf (float) : Final temperature related to the dynamic of Metropolis

alpha (float) : Geometric factor to decrease Temperature (0 < alpha < 1)

n_classes (array) : Number of classes for the displayed histogram (ex : range(100))

t_iter (int) : Number of iterations for a Markov chain

n_iter (int) : Number of Markov chains generated for each Temperature

Returns: G_out (matrix) : Output state of the dynamic of Metropolis

E_out (float) : Energy associated to the new state G_out

freqs (array) : Frequencies associated to each energy

CHAPTER

TWO

UTILS

2.1 Change of coordinates

`utils.coc.`**`cart2kep`**(*R*, *V*, *mu*)

Converts coordinates of a body from its cartesian coordinates into its orbitals elements (coe)

Arguments :

R (array) : Position of the body in the ECI frame

V (array) : Velocity of the body in the ECI frame

mu (float) : Characteristic parameter of the central body

Returns [] coe (array) : Body’s orbital elements (SMA, ECC, INC, AOP, RAAN, TA)

`utils.coc.`**`kep2cart`**(*coe*, *mu*)

Converts coordinates of a body from orbital elements (coe) into cartesian coordinates Arguments :

coe (array) : Orbital elements of the body (SMA, ECC, INC, AOP, RAAN, TA)

mu (float) : Characteristic parameter of the central body

Returns [] r (array) : Concatenation of the body’s position and velocity (X, Y, Z, VX, VY, VZ) in the geocentric frame

2.2 Useful constants

2.3 Loading debris data

Created on Wed Nov 24 10:19:33 2021

@author: g.pierre

`utils.debris_data_loader.`**`convertTLEtoDF`**(*TLE_String*)

Function which aims to convert data in TLE format towards pandas dataframe (for our set of debris)

Arguments :

TLE_String (string) : Concatenated TLEs (string) of each object

Returns :

TLE_DF (pandas dataframe) : DataFrame containing orbital parameters, time and mass for each debris `utils.debris_data_loader.`**`recoveringDebrisData`**`(*args)`

Function which aims to recover data from orbiting objects from Space-Track.org

Arguments :

args (list of int) : Norad IDs of objects of interest, default value is the list given in constant.py

Returns :

TLE_String (string) : Concatenated TLEs (string) of each object

2.4 General calculus

This module computes intermadiary data like velocity, angular momentum, effect of J2 perturbation on RAAN

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

u

INDEX