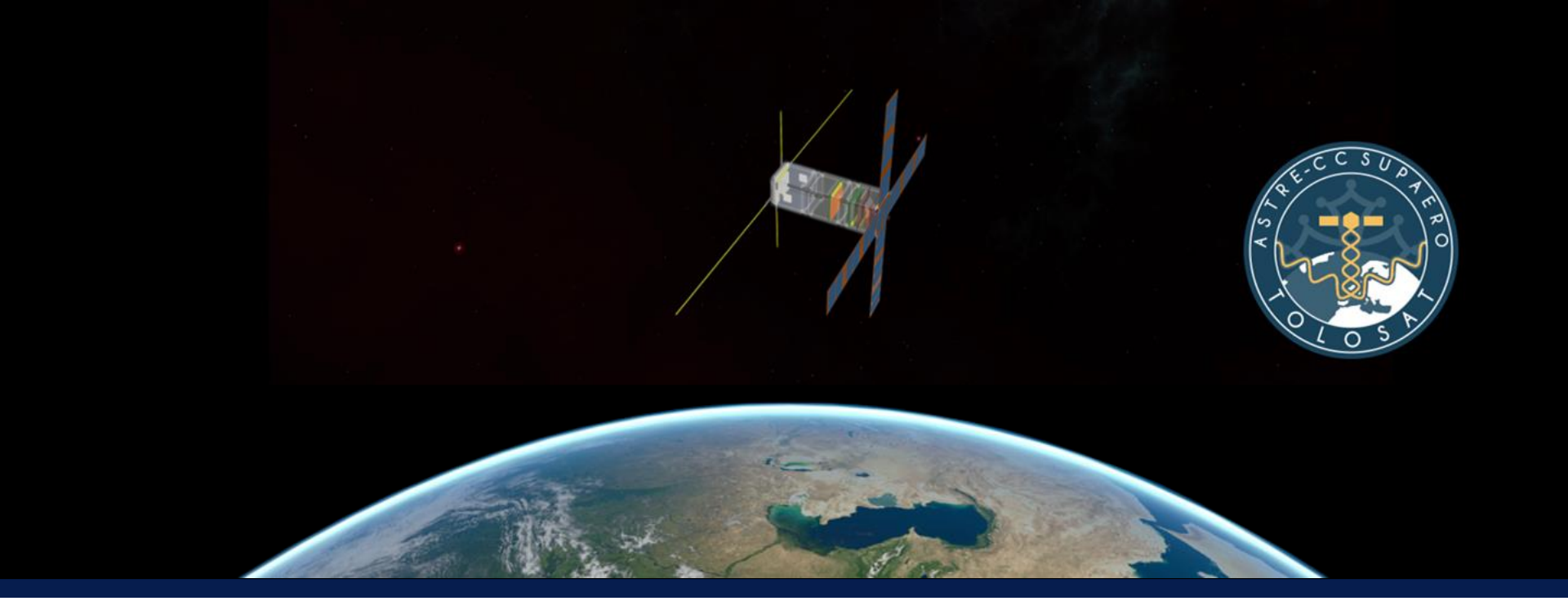


Mission Analysis in Python

An introduction to Tudatpy

Arnaud MULLER

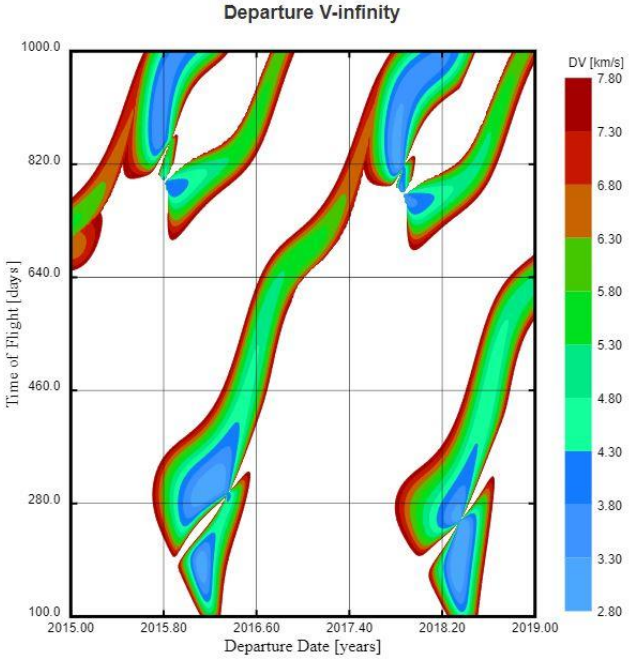
Table of Contents

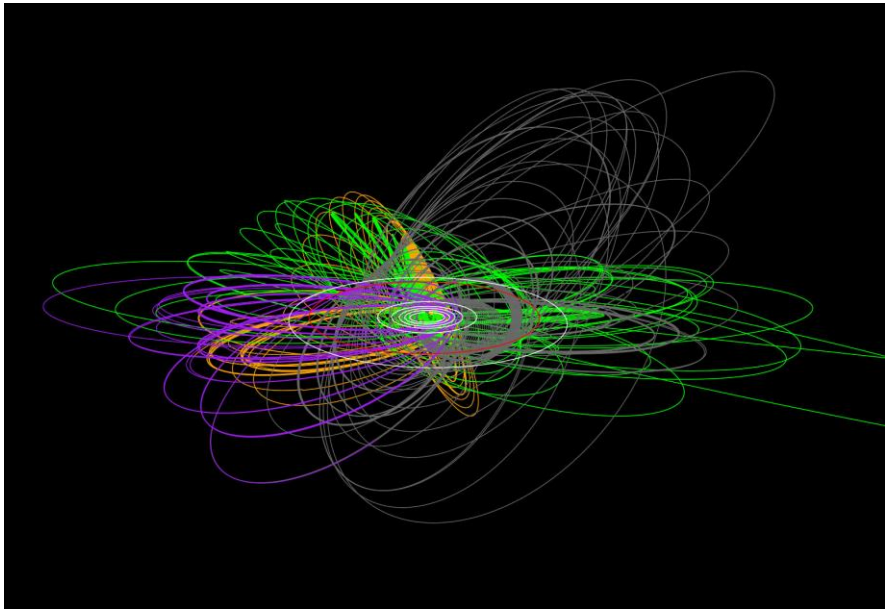
Introduction

Numerical Astrodynamics

- Solving differential equations using a numerical integrator
- Choosing which equations & models to use
- Interpreting, verifying and applying the results

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, t; \mathbf{p}\right) \rightarrow \mathbf{x}(t_i)$$





What is Tudat ?

Original project rationale

Validated and modular tool for TU Delft MSc and PhD students doing astrodynamics research

- ***Validated:*** all code verified by unit tests
- ***Modular:*** flexible setup for diverse applications (re-entry, ascent, transfer design, orbit optimization, …)
- **Prevent people from reinventing the wheel**

→ **As an open-source project, has become a more widely-used “software lab” for astrodynamics**

tudat

ad astra per aspera

What is Tudatpy ?

The best way to interact with Tudat

Tudat: core of the software written in C++, performs all the computations

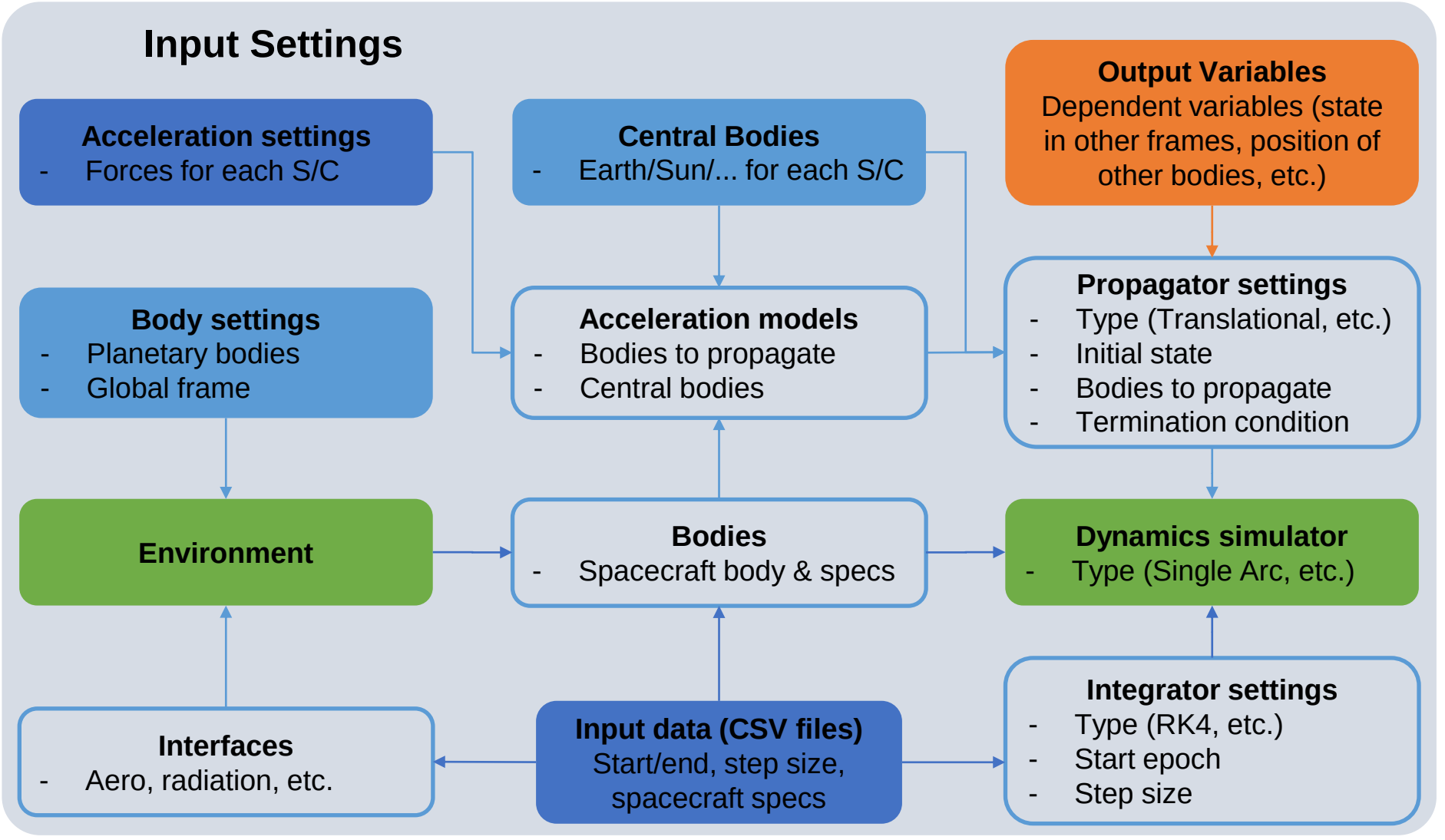
Tudatpy: user-friendly Python interface, keeps the performance of C++

Plotting and analysis of the results can be done directly in Python

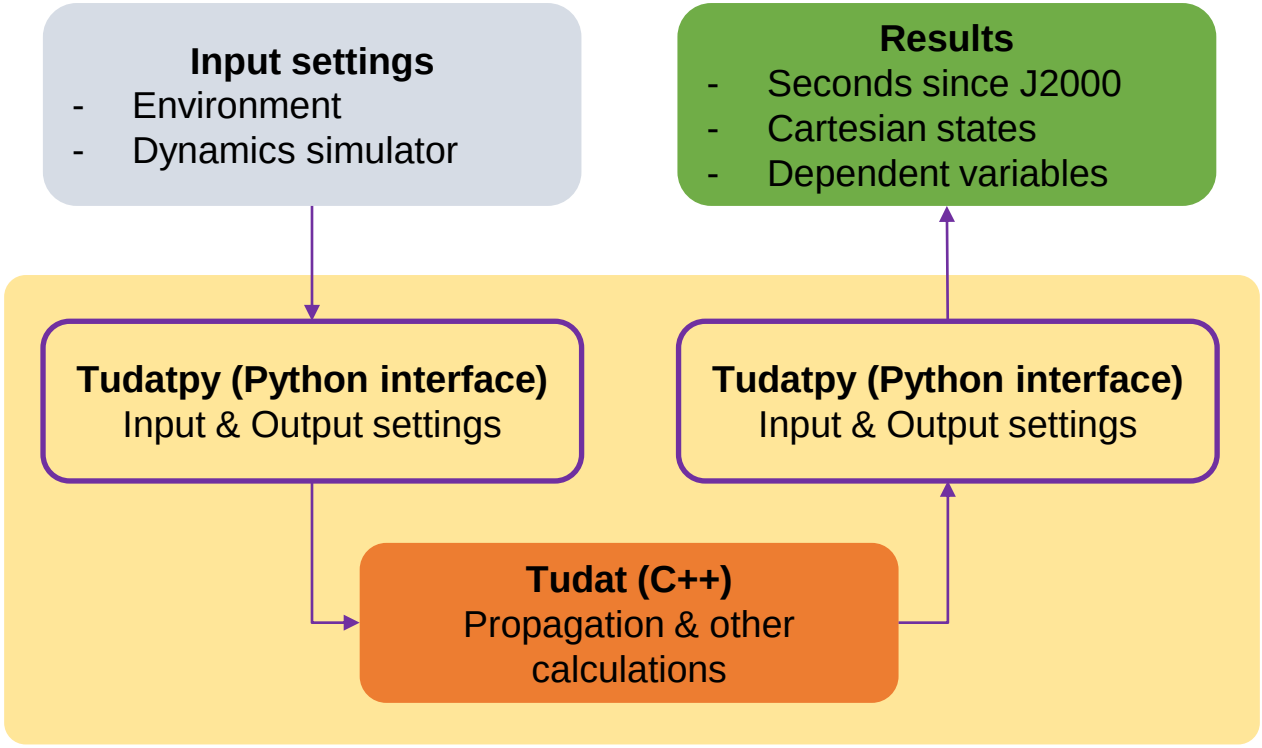
tudat

ad astra per aspera

Tudatpy Workflow



Tudatpy Workflow



Coding with Tudatpy

TOLOSAT

Select input data

```
# Initial settings (independent of tudat)
orbit_name = 'SSO6'
dates_name = '5days'
spacecraft_name = 'Tolosat'
groundstation_name = 'test_station'
```

Set start & end dates

```
# Set simulation start and end epochs (in seconds since J2000 = January 1, 2000 at 00:00:00)
dates = get_dates(dates_name)
simulation_start_epoch = time_conversion.julian_day_to_seconds_since_epoch(
    time_conversion.calendar_date_to_julian_day(dates["start_date"]))
simulation_end_epoch = time_conversion.julian_day_to_seconds_since_epoch(
    time_conversion.calendar_date_to_julian_day(dates["end_date"]))
```

Coding with Tudatpy

TOLOSAT

Create planetary bodies & set frame

```
# Create default body settings and bodies system
bodies_to_create = ["Earth", "Sun", "Moon", "Jupiter"]
global_frame_origin = "Earth"
global_frame_orientation = "J2000"
body_settings = environment_setup.get_default_body_settings(
    bodies_to_create, global_frame_origin, global_frame_orientation)
bodies = environment_setup.create_system_of_bodies(body_settings)
```

Add spacecraft

```
# Add vehicle object to system of bodies
bodies.create_empty_body("Spacecraft")
bodies.get("Spacecraft").mass = get_spacecraft(spacecraft_name)["mass"]
```

Define bodies to propagate with respect to which body

```
# Define bodies that are propagated and their respective central bodies
bodies_to_propagate = ["Spacecraft"]
central_bodies = ["Earth"]
```

Coding with Tudatpy

Create aerodynamic & radiation interfaces

```
# Create aerodynamic coefficient interface settings, and add to vehicle
reference_area = get_spacecraft(spacecraft_name)["drag_area"]
drag_coefficient = get_spacecraft(spacecraft_name)["drag_coefficient"]
aero_coefficient_settings = environment_setup.aerodynamic_coefficients.constant(
    reference_area, [drag_coefficient, 0, 0]
)
environment_setup.add_aerodynamic_coefficient_interface(
    bodies, "Spacecraft", aero_coefficient_settings)

# Create radiation pressure settings, and add to vehicle
reference_area_radiation = get_spacecraft(spacecraft_name)["srp_area"]
radiation_pressure_coefficient = get_spacecraft(spacecraft_name)["reflectivity_coefficient"]
occulting_bodies = ["Earth"]
radiation_pressure_settings = environment_setup.radiation_pressure.cannonball(
    "Sun", reference_area_radiation, radiation_pressure_coefficient, occulting_bodies
)
environment_setup.add_radiation_pressure_interface(
    bodies, "Spacecraft", radiation_pressure_settings)
```

Coding with Tudatpy

TOLOSAT

Choose acceleration settings

```python
# Define accelerations acting on the spacecraft
acceleration_settings_spacecraft = dict(
    Sun=[
        propagation_setup.acceleration.cannonball_radiation_pressure(),
        propagation_setup.acceleration.point_mass_gravity()
    ],
    Earth=[
        propagation_setup.acceleration.spherical_harmonic_gravity(10, 10),
        propagation_setup.acceleration.aerodynamic()
    ],
    Moon=[
        propagation_setup.acceleration.point_mass_gravity()
    ],
    Jupiter=[
        propagation_setup.acceleration.point_mass_gravity()
    ]
)

acceleration_settings = {"Spacecraft": acceleration_settings_spacecraft}

# Create acceleration models
acceleration_models = propagation_setup.create_acceleration_models(
    bodies, acceleration_settings, bodies_to_propagate, central_bodies
)
```

Coding with Tudatpy

TOLOSAT

Set initial state of the spacecraft

```python
# Set initial conditions for the satellite
earth_gravitational_parameter = bodies.get("Earth").gravitational_parameter
orbit = get_orbit(orbit_name)
initial_state = element_conversion.keplerian_to_cartesian_elementwise(
    gravitational_parameter=earth_gravitational_parameter,
    semi_major_axis=orbit["semi_major_axis"],
    eccentricity=orbit["eccentricity"],
    inclination=np.deg2rad(orbit["inclination"]),
    argument_of_periapsis=np.deg2rad(orbit["argument_of_periapsis"]),
    longitude_of_ascending_node=np.deg2rad(orbit["longitude_of_ascending_node"]),
    true_anomaly=np.deg2rad(orbit["true_anomaly"]),
)
```

Choose dependent variables to export (on top of cartesian states)

```python
# Setup dependent variables to be save
sun_position_dep_var = propagation_setup.dependent_variable.relative_position("Sun", "Earth")
earth_position_dep_var = propagation_setup.dependent_variable.relative_position("Earth", "Earth")
keplerian_states_dep_var = propagation_setup.dependent_variable.keplerian_state("Spacecraft", "Earth")
ecef_pos_dep_var = propagation_setup.dependent_variable.central_body_fixed_cartesian_position("Spacecraft", "Earth")
dependent_variables_to_save = [sun_position_dep_var, earth_position_dep_var, keplerian_states_dep_var, ecef_pos_dep_var]
```

Coding with Tudatpy

TOLOSAT

Create termination condition

```
# Create termination settings
termination_condition = propagation_setup.propagator.time_termination(simulation_end_epoch)
```

Create propagator & integrator settings

```
# Create propagation settings
propagator_settings = propagation_setup.propagator.translational(
    central_bodies,
    acceleration_models,
    bodies_to_propagate,
    initial_state,
    termination_condition,
    output_variables=dependent_variables_to_save
)

# Create numerical integrator settings
fixed_step_size = dates["step_size"].total_seconds()
integrator_settings = propagation_setup.integrator.runge_kutta_4(
    simulation_start_epoch, fixed_step_size
)
```

Coding with Tudatpy

TOLOSAT

Assemble dynamics simulator → STARTS PROPAGATION

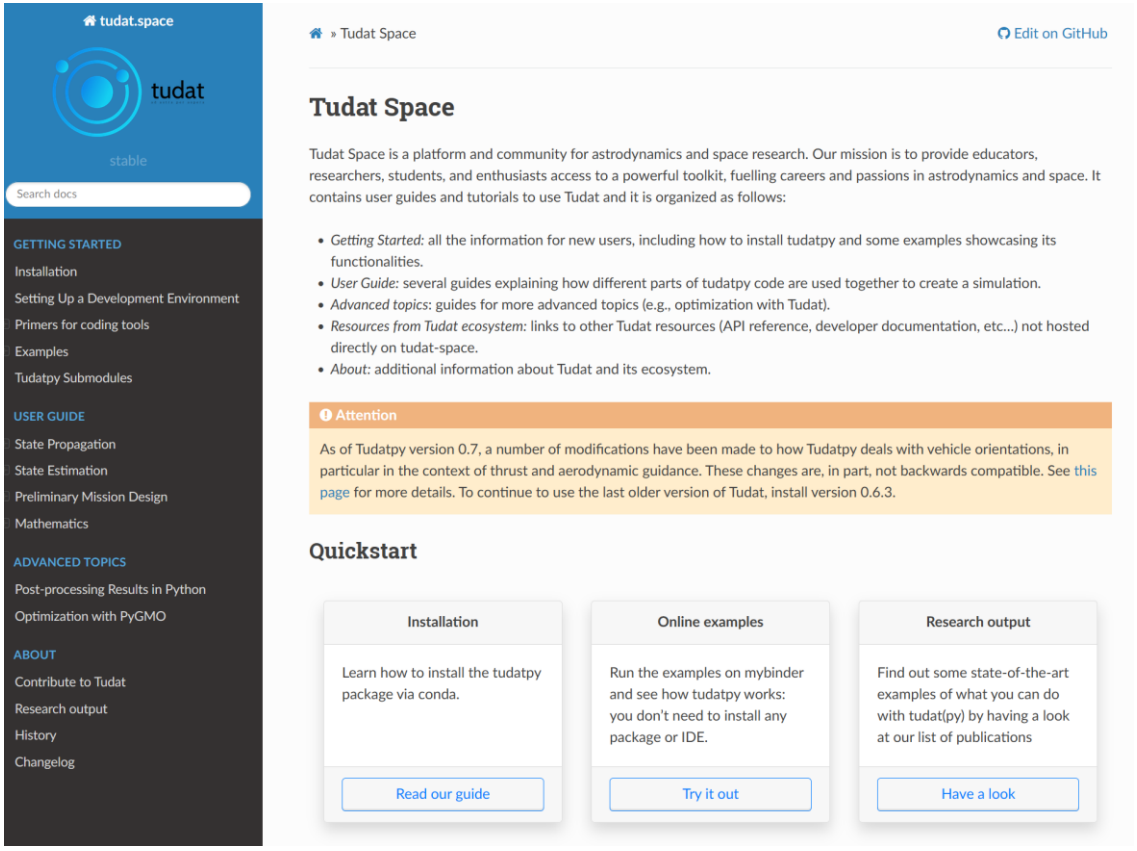
```
# Create simulation object and propagate the dynamics
dynamics_simulator = numerical_simulation.SingleArcSimulator(
    bodies, integrator_settings, propagator_settings
)
```

Extract results

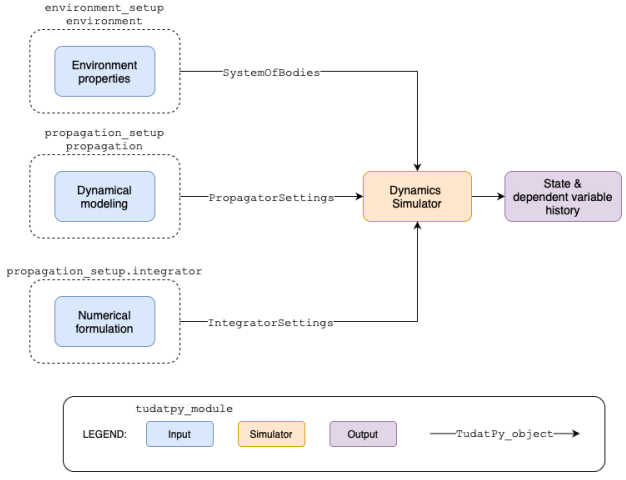
```
# Extract the resulting state history and convert it to a ndarray
states = dynamics_simulator.state_history
states_array = result2array(states)
dependent_variables_history = dynamics_simulator.dependent_variable_history
dependent_variables_history_array = result2array(dependent_variables_history)

sun_radius = bodies.get("Sun").shape_model.average_radius
earth_radius = bodies.get("Earth").shape_model.average_radius
states_array[:, 0] = states_array[:, 0] - states_array[0, 0]
satellite_position = states_array[:, 1:4]
sun_position = dependent_variables_history_array[:, 1:4]
earth_position = dependent_variables_history_array[:, 4:7]
keplerian_states = dependent_variables_history_array[:, 7:13]
ecef_position = dependent_variables_history_array[:, 13:16]
```

Using the documentation

Tudat documentation

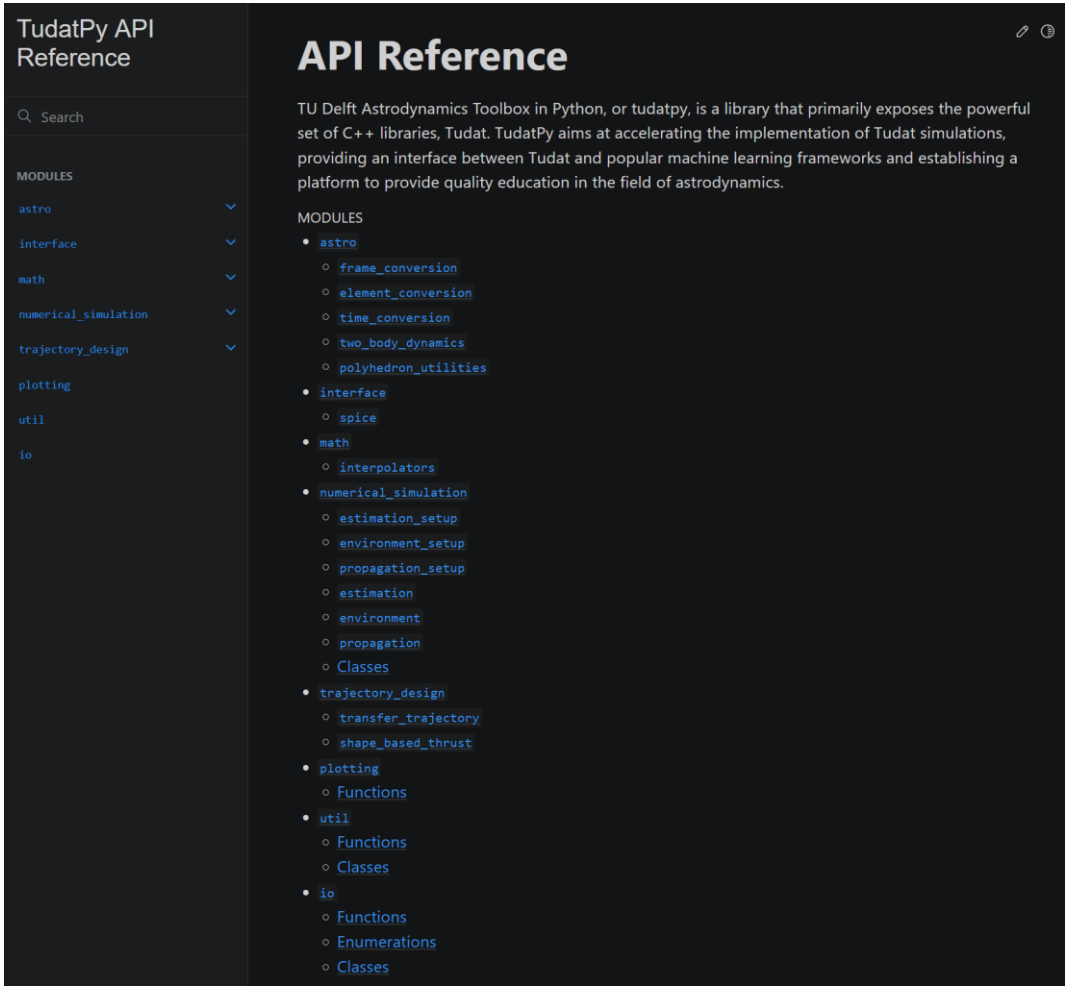


Useful to understand the intricacies of Tudat and find code snippets.

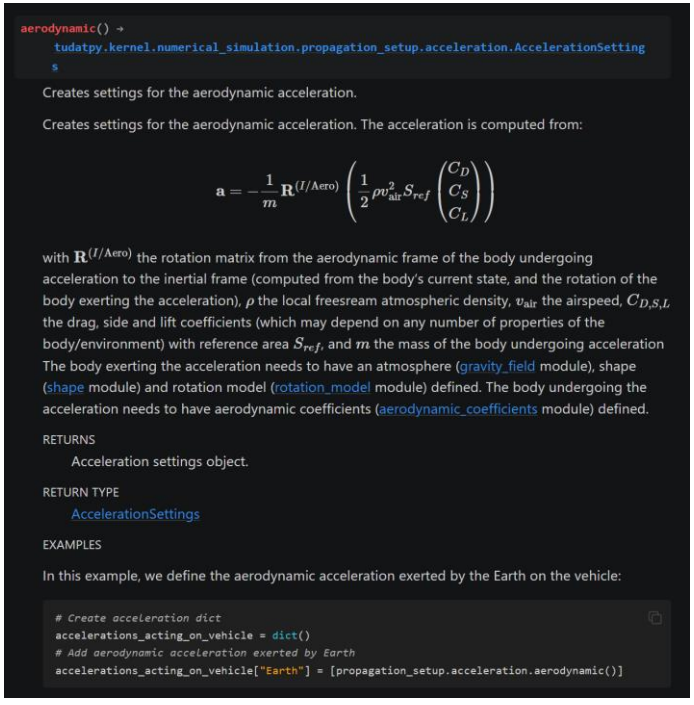


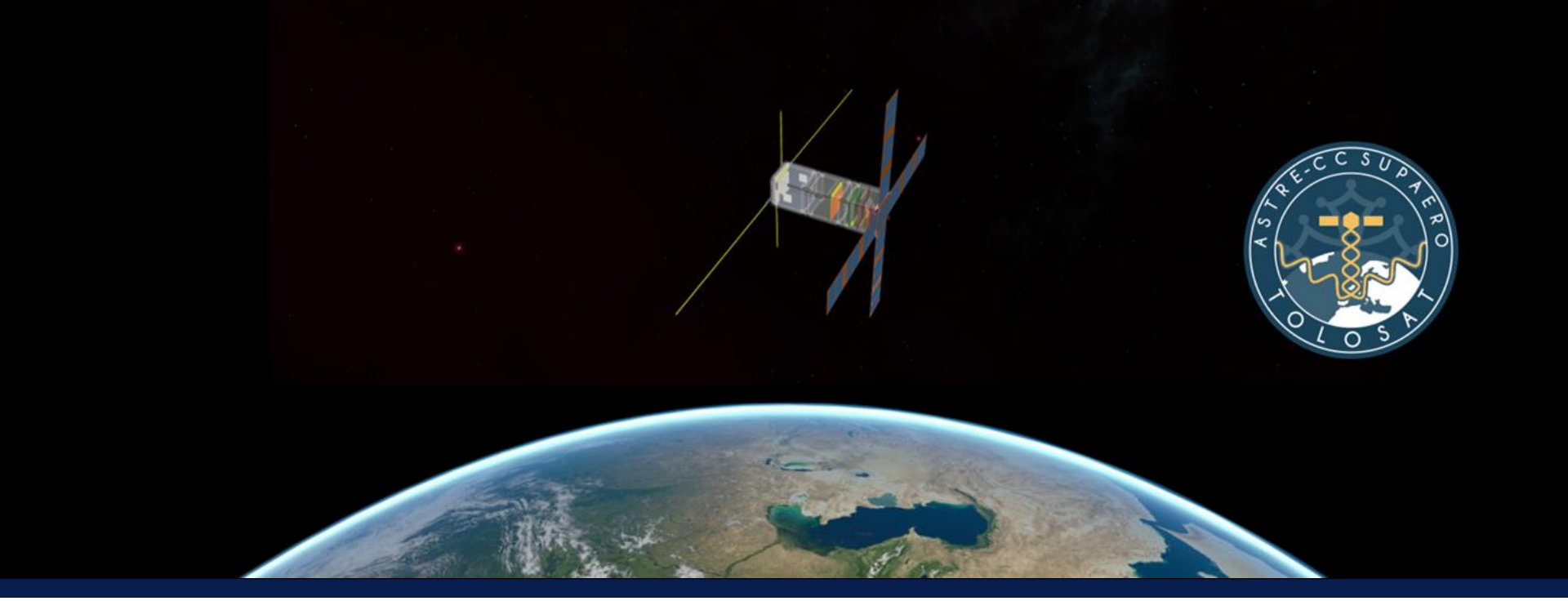
Using the documentation

Tudatpy API reference



Useful to see exactly how to use the Tudatpy interface and what features are available.





References & Links

Dominic Dirkx, AE4868 Numerical Astrodynamics, TU Delft (D.Dirkx@tudelft.nl)

TU Delft Astrodynamics Toolbox (TUDAT) https://docs.tudat.space/en/stable/

Tudat documentation https://docs.tudat.space/en/stable/

Tudatpy API reference https://py.api.tudat.space/en/latest/