

Tutorials 4.0 Propagator

De Wiki

Aller à : [navigation](#), [rechercher](#)

[AOLEvent](#) > [User Manual 4.13 Events detection: Presentation](#) > [Utilisateur:3.149.233.72](#) > [Spécial:Pages liées/Tutorials 4.0 Dates](#) > [Tutorials 4.0 Propagator](#)

Sommaire

- [1 What is a propagator?](#)
- [2 With a step handler](#)
- [3 With a STOP event](#)
- [4 With attitude laws](#)
- [5 With a mass model](#)
- [6 Using force models](#)
 - [6.1 Using a potential field](#)
 - [6.2 Using a drag model](#)
 - [6.3 Using a drag and lift model](#)
- [7 With maneuvers](#)
 - [7.1 Impulsive maneuvers](#)
 - [7.2 Continuous maneuvers](#)
 - [7.3 Sequence of maneuvers](#)

What is a propagator?

We must differentiate a numerical **integrator** and a numerical **propagator** ...

A numerical integrator is a low level block allowing to integrate differential equations, not only those linked to a trajectory. So, if you want to use this integrator level, you will have to define your own equations, specially the second term which can be complex when you have to consider a lot of perturbation terms in a trajectory extrapolation.

That is why **PATRIUS** proposes a higher level object allowing to propagate an orbit (and more generally a trajectory). This object already have internally some differential kind of equations (**cartesian** form, **equinoctial** form, ...) and has some very powerful mechanism to add **perturbation force models** as we could see in further tutorials by using the [addForceModel\(\)](#) method.

The basic philosophy is then:

1. to create a numerical propagator giving it an associated numerical integrator
2. to give it initial conditions via a [SpacecraftState](#) object (including at least an orbit but also possibly a mass model and an attitude).
3. optionally to specify which kind of equations and which frame to use (by default the ones consistent with the initial orbit definition).
4. at last to call for the [propagate\(\)](#) method up to a certain absolute date. It will return a new [SpacecraftState](#) object.

Code examples:

- [Numerical propagation using a 4th order Runge Kutta integrator](#)
- [Numerical propagation using a DOP integrator](#)

With a step handler

A step handler (here a fixed step handler) is the functionality allowing to extract some data along the numerical propagation. This mechanism is done thanks to an internal interpolation and do not modify the precision of the numerical integration.

For example, if you have a **RK4** integrator with a time integration step of 10s and if you want to extract data every 5s, it will not interfere on the 10s steps (meaning that it will not reduce the time integration step to 5s).

Code example: [Numerical propagation with a fixed step handler](#)

With a STOP event

Thanks to a "G-stop" functionality included into numerical integrators, it is possible to add events to the propagator. This one will then automatically detect when the event occurred and will execute the action associated with the event (a STOP action in the first example below). A large amount of predefined events are already available with **PATRIUS**. Here, the example will treat of an altitude event. The second example is more or less the same kind of example than the previous one but, in that case, we added a custom event. This is done using an internal class in order to get all in the same method but, practically, it is recommended to define this custom event in a specific class for a better readability.

Code examples:

- [Numerical propagation with a STOP event](#)
- [Numerical propagation with a custom event](#)

With attitude laws

The way to consider a single attitude law is relatively easy as we have just to create an attitude law (in the code below, a **LOF** law) and call for the `setAttitudeProvider()` method. For more explanations about creating attitude laws, see [specific tutorials](#).

To add an attitude sequence in a numerical propagator is just a bit more complex than just adding a single attitude law:

- first, we have to define an attitude sequence. In the code below, we considered a sequence only with two switches and two laws but it can be actually more sophisticated. For more explanations about the way to build attitude sequences, see [specific tutorials](#).
- then, we must add the sequence to the propagator by using the `setAttitudeProvider()` method.

```
propagator.setAttitudeProvider(seqAtt);
```

- at last, we must not forget to tell to the sequence that a propagator is "looking" at it by calling the `registerSwitchEvents()` method

```
seqAtt.registerSwitchEvents(propagator);
```

Code examples:

- [Numerical propagation with an attitude law](#)
- [Numerical propagation with an attitude sequence](#)

With a mass model

A mass model allows, within a propagation, to take into account mass variations (typically due to maneuvers). This is done thanks to the [Assembly](#) construction (see also specific tutorials on it). Anyway, when using such [MassModel](#), it is mandatory :

- to add it to the initial `SpacecraftState`
- to feed the additional equations to the propagator by using the ([setMassProviderEquation\(\)](#)) method.

Code example: [Numerical propagation with a mass model](#)

Using force models

If nothing specific is requested, a numerical propagator will only deal with a Keplerian motion using the central term of the potential got from the initial orbit. If we want to add other force models, we will have to use the [addForceModel\(\)](#) method.

Using a potential field

In the first example below, we will give the way to use a [ICGEM](#) potential field with [Droziner](#) equations up to 8x8 degrees. Up to now, it is not possible simply to select a specific potential field if several ones are available. For such functionality, we can use the utility method given at the end of this tutorial.

Code example: [Numerical propagation with a specific potential field](#)

Using a drag model

In the second example, we will take into account a drag model and have to do both things:

- adding to the vehicle (i.e. the [Assembly](#)) the [AeroSphereProperty](#)
- creating a force model including this aerodynamic drag model plus an atmospheric model and adding it to the propagator

Code example: [Numerical propagation with a drag model](#)

Using a drag and lift model

A bit like the previous example, we are going to see how to use a drag and a lift model. The way to do it is almost the same:

- adding to the vehicle (i.e. the [Assembly](#)) the [AeroGlobalProperty](#) (rather than the [AeroSphereProperty](#))
- creating a force model including this aerodynamic lift+drag model plus an atmospheric model and

adding it to the propagator

Nevertheless, another important point is mandatory: as we need to know which is the direction of the lift (perpendicular to the drag which is opposite to the relative velocity), we must add an attitude law to the Spacecraft as it is done below by using the [setAttitudeProvider\(\)](#) method.

The only difference between both examples is due to the change of the atmospheric model.

Code examples:

- [Numerical propagation with a drag and lift model](#)
- [Numerical propagation with a drag and lift model and the MSIS2000 atmospheric model](#)

With maneuvers

Impulsive maneuvers

Here we consider we will just add impulsive maneuvers. A way to do it is to consider such maneuvers as events. So, we will use the [addEventDetector\(\)](#) method.

Nevertheless, a very important point to consider is the fact that, due to a maneuver, the mass of the spacecraft will be modified: so, it is mandatory to define a [SpacecraftState](#) with a [MassModel](#) (see [specific tutorial](#)).

Code example: [Numerical propagation with an impulsive maneuver](#)

Continuous maneuvers

Here we will not use the principle of adding an event to the propagator but adding a force model, considering the thrust is a supplementary force. To do that, we will use the [addForceModel\(\)](#) method. For more details in the way to define a continuous thrust maneuver, see the specific items of the user manual, the JavaDoc or specific tutorials.

As for the previous tutorial about impulsive maneuvers, it is mandatory to define a [SpacecraftState](#) with a [MassModel](#).

At last, if the thrust direction is defined in the vehicle frame, it is also mandatory to define an attitude law (or a sequence) to be able to know the actual direction of the thrust in the integration frame.

Code example: [Numerical propagation with a continuous maneuver](#)

Sequence of maneuvers

Here we will use a sequence of maneuvers. This sequence will include both impulsive and continuous maneuvers (see specific tutorials to know how to do it). The main point to know is the fact that, once the maneuver sequence has been built, we do not have to pass it to the propagator ... but we must pass the propagator to the sequence using the [applyTo\(\)](#) method !

And, as for the two previous tutorials, do not forget to define a [MassModel](#) and eventually an attitude law ...

Code example: [Numerical propagation with a sequence of maneuvers](#)

Récupérée de « http://patrius.cnes.fr/index.php?title=Tutorials_4.0_Propagator&oldid=2713 »
Catégorie :

- [Tutorials 4.0](#)

Menu de navigation

Outils personnels

- [3.136.233.222](#)
- [Discussion avec cette adresse IP](#)
- [Créer un compte](#)
- [Se connecter](#)

Espaces de noms

- [Page](#)
- [Discussion](#)

Variantes

Affichages

- [Lire](#)
- [Voir le texte source](#)
- [Historique](#)
- [Exporter en PDF](#)

Plus

Rechercher

<input type="text"/>	<input type="button" value="Rechercher"/>	<input type="button" value="Lire"/>
----------------------	---	-------------------------------------

PATRIUS

- [Welcome](#)

Evolutions

- [Main differences between V4.13 and V4.12](#)
- [Main differences between V4.12 and V4.11](#)
- [Main differences between V4.11 and V4.10](#)
- [Main differences between V4.10 and V4.9](#)
- [Main differences between V4.9 and V4.8](#)
- [Main differences between V4.8 and V4.7](#)
- [Main differences between V4.7 and V4.6.1](#)
- [Main differences between V4.6.1 and V4.5.1](#)
- [Main differences between V4.5.1 and V4.4](#)
- [Main differences between V4.4 and V4.3](#)
- [Main differences between V4.3 and V4.2](#)
- [Main differences between V4.2 and V4.1.1](#)
- [Main differences between V4.1.1 and V4.1](#)
- [Main differences between V4.1 and V4.0](#)
- [Main differences between V4.0 and V3.4.1](#)

User Manual

- [User Manual 4.13](#)
- [User Manual 4.12](#)
- [User Manual 4.11](#)
- [User Manual 4.10](#)
- [User Manual 4.9](#)
- [User Manual 4.8](#)
- [User Manual 4.7](#)
- [User Manual 4.6.1](#)
- [User Manual 4.5.1](#)
- [User Manual 4.4](#)
- [User Manual 4.3](#)
- [User Manual 4.2](#)
- [User Manual 4.1](#)
- [User Manual 4.0](#)
- [User Manual 3.4.1](#)
- [User Manual 3.3](#)

Tutorials

- [Tutorials 4.5.1](#)
- [Tutorials 4.4](#)
- [Tutorials 4.1](#)
- [Tutorials 4.0](#)

Links

- [CNES freeware server](#)

Navigation

- [Accueil](#)
- [Modifications récentes](#)
- [Page au hasard](#)
- [Aide](#)

Outils

- [Pages liées](#)
- [Suivi des pages liées](#)
- [Pages spéciales](#)
- [Adresse de cette version](#)
- [Information sur la page](#)
- [Citer cette page](#)

- Dernière modification de cette page le 17 août 2020 à 09:14.
- [Politique de confidentialité](#)
- [À propos de Wiki](#)
- [Avertissements](#)

- 