User Manual 4.14 Multi Propagation

De Wiki
Aller à : navigation, rechercher
User Manual 4.14 Multi Propagation

Sommaire

Introduction

Scope

This section describes the multi propagator provided by the Patrius library. As of version 4.14, both numerical and analytical propagators are available for multi spacecraft propagation. Generic features about propagators as well as other type of propagators are detailed [ORB_PGEN_Home here].

Javadoc

All the classes related to numerical propagation are in the

`fr.cnes.sirius.patrius.propagation.numerical.multi` package of the Patrius library and those related to analytical propagation are in the `fr.cnes.sirius.patrius.propagation.analytical.multi package`. The classes related to events detection are in the package `fr.cnes.sirius.patrius.events.detectors` of the Patrius library. However, the dedicated interface for multi event detectors is in `fr.cnes.sirius.patrius.events` package of the Patrius library. All the classes related to attitude providers for multi satellites are in the package `fr.cnes.sirius.patrius.attitudes.multi`.

Library	Javadoc
Patrius	Package fr.cnes.sirius.patrius.events
Patrius	Package fr.cnes.sirius.patrius.events.utils
Patrius	Package fr.cnes.sirius.patrius.propagation.numerical.multi
Patrius	Package fr.cnes.sirius.patrius.propagation.precomputed.multi
Patrius	Package fr.cnes.sirius.patrius.propagation.sampling.multi
Patrius	Package fr.cnes.sirius.patrius.propagation.analytical.multi
Patrius	Package fr.cnes.sirius.patrius.attitudes.multi

Links

Other useful links can be found here :

- [ORB_PRO_Home Classical propagation chapter]
- [FDY_SST_Home SpacecraftState chapter]

Useful Documents

None as of now.

Package Overview

Multi Numerical Propagator

The multi numerical propagator architecture is copied from Propagator interface and NumericalPropagator class, and added to PATRIUS library.

Multi Analytical Propagator

The multi analytical propagator architecture is copied from the MultiPropagator interface and the AbstractPropagator class, and added to the PATRIUS library.

Three propagation modes

Like single spacecraft propagation, the multi numerical spacecraft propagation can be performed in different propagation modes : slave, master and ephemeris generation. The ephemeris mode is however not available in the case of multi analytical propagation (not needed as of version 4.14).

Features Description

The PATRIUS library offers a multi numerical propagator and a multi analytical propagator. It aims at propagating several SpacecraftState at the same time using either numerical or analytical propagation (or both under certain conditions). The N SpacecraftState are propagated during the same time interval

- using the same FirstOrderIntegrator, in the numerical case;
- using mono-propagators like SpacecraftStateProviders to compute the SpacecraftStates at the desired date, in the analytical case.

Each state or mono-propagator is identified with an ID of type String. All states can be completely different but they should have the same internal date. This is particularly important for numerical propagation, because the user directly adds timestamped SpacecraftStates to be propagated. In the case of analytical propagation, the user only chooses a date that is used to compute all the initial states.

In numerical case, as for the mono-satellite propagation, it is possible to define the following global propagation parameters:

- The type of orbital parameters and position angle
- The propagation mode

For each state, it is possible to configure the following elements:

- The central attraction coefficient
- The force models
- The tolerances used for orbital parameters and the tolerances applied on additional states.
- The additional states equations associated with the additional states
- The attitude providers

The analytical case is simpler, the propagation mode is the only global parameter that can be set. Moreover, only attitude providers are used for each state.

In both cases, it is possible to define event detectors applied on a specific state or global event detectors applied on several states.

Note that the NumericalPropagator and the MultiNumericalPropagator do not use anymore the Newtonian gravity model by default. It should now be added manually to the list of the force models before starting the propagation.

Getting started with multi numerical propagation

Here is presented a basic instantiation of the multi numerical propagator.

Create propagator from FirstOrderIntegrator

The MultiNumericalPropagator should be created by giving a FirstOrderIntegrator.

```
final MultiNumericalPropagator propagator = new
MultiNumericalPropagator(integrator);
```

The integrator could be declared using absolute and relative tolerances represented by :

- a scalar value

```
final double abstolScal = 1.0e-10;
final double reltolScal = 1.0e-10;
final FirstOrderIntegrator integratorScal = new
DormandPrince853Integrator(0.001, 200, abstolScal, reltolScal);
```

- a vector

```
final double[] abstolVec = { 1e-5, 1e-5, 1e-5, 1e-8, 1e-8, 1e-8};
final double[] reltolVec = { 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10};
final FirstOrderIntegrator integratorVec = new
DormandPrince853Integrator(0.1, 60., abstolVec, reltolVec);
```

The orbit tolerances (vector or scalar tolerances) given to the integrator are used as default tolerances. The orbit tolerances could be defined in vector type using the method ***setOrbitTolerance***.

Add initial states

Initial states could be added to the propagator using the method ***addInitialState(SpacecraftState, String)***. Each added state is defined by a unique ID. An error is raised if :

- the input spacecraft ID is empty or null.
- a state with the same ID was already added to the propagator.
- the date associated with the added state is different from the date associated with the states previously added. (All states are propagated from the same date).

Note that, the initial state should be added before adding an additional equation, an attitude provider, a force model, an event ... etc ... associated with this state. Otherwise, the input spacecraft Id will not be recognized.

The map of the added states could be retrieved with ***getInitialStates()***

Add additional equations

As for the NumericalPropagator, additional equations corresponding to the additional states of each states can be added to the propagator.

These additional equations are given to the multi numerical propagator for a specific state thanks to the following methods :

- ***addAdditionalEquation(AdditionalEquation, String)***
- ***addAttitudeEquation(AttitudeEquation, String)***
- ***setMassProviderEquation(MassProvider, String)*** (This method should be called only once, mass provider must be mass provider used in force models)

The method ***setAdditionalStateTolerance(String, double[], double[])*** may be used to add variation tolerance values (absolute and relative) to a specific additional state.

For each state, the additional states should correspond with the added additional equations (same name, same size). As of PATRIUS 4.13, partial derivatives equations are available for multi-numerical propagation with the class `MultiPartialDerivativesEquations`. They are used exactly the same way as for mono-satellite propagation. Note that a set of partial derivatives equations has to be provided for each satellite, allowing for different sets of partial derivatives equations for each satellite.

See [ORB_PRO_UseP mono numerical propagation] chapter for more details.

The following example shows how to propagate two basic SpacecraftState :

```
// Initial date
final AbsoluteDate date = AbsoluteDate.J2000_EPOCH;

// Constants
final double mu = Constants.EGM96_EARTH_MU;
final Frame gcrf = FramesFactory.getGCRF();
final String STATE1 = "STATE1";
final String STATE2 = "STATE2";
final String BODY = "body";

// First initial state
final Orbit orbit1 = new KeplerianOrbit(7500000, 0.001, 0.40, 0, 0,
0,PositionAngle.MEAN, gcrf, date, mu);
final MassProvider massModel = new SimpleMassModel(1000., BODY);
final SpacecraftState state1 = new SpacecraftState(orbit1, massModel);

// Second initial state
final Vector3D position = new Vector3D(7.0e6, 1.0e6, 4.0e6);
final Vector3D velocity = new Vector3D(-500.0, 8000.0, 1000.0);
final Orbit orbit2 = new EquinoctialOrbit(new PVCoordinates(position,
velocity), gcrf, date, mu);
final AttitudeProvider law = new
ConstantAttitudeLaw(FramesFactory.getEME2000(), Rotation.IDENTITY);
final Attitude attitude = law.getAttitude(orbit2, date, gcrf);
final SpacecraftState state2 = new SpacecraftState(orbit2, attitude);

// Add initial states to the propagator
propagator.addInitialState(state1, STATE1);
propagator.addInitialState(state2, STATE2);

// Note that the NumericalPropagator and the MultiNumericalPropagator do not
use anymore the Newtonian gravity model by default. It should now be added
manually to the list of the force models before starting the propagation.
mainPropagator.addForceModel(new DirectBodyAttraction(new
NewtonianGravityModel(mu, STATE1));
mainPropagator.addForceModel(new DirectBodyAttraction(new
NewtonianGravityModel(mu, STATE2));

// Add additional equation associated with the mass model of the first state
// Mass provider must be mass provider used in force models.
```

```
propagator.setMassProviderEquation(massModel, STATE1);
propagator.setAdditionalStateTolerance("MASS_" + BODY, new double[]{1e-7},
new double[]{1e-7}, STATE1);

// Set attitude provider associated with the second state
propagator.setAttitudeProvider(law, STATE2);

// propagation
final Map<String, SpacecraftState> finalStates =
propagator.propagate(date.shiftedBy(propagationDuration));
```

Propagate the attitude

As for the NumericalPropagator, to propagate the Attitude of a specific state, two treatments could be applied :

- compute the attitude with an AttitudeProvider : ***setAttitudeProvider(AttitudeProvider, String)***.
  It is possible to deals with :
  - a single attitude by calling ***setAttitudeProvider(AttitudeProvider, String)***
  - two attitudes by calling ***setAttitudeProviderForces(AttitudeProvider, String)*** or ***setAttitudeProviderEvents(AttitudeProvider, String)***. It is not possible to call ***setAttitudeProvider(AttitudeProvider, String)*** and ***setAttitudeProviderForces(AttitudeProvider, String)*** (or ***setAttitudeProviderEvents(AttitudeProvider, String)***)
- propagate the attitude as a 7-dimension additional state : ***addAttitudeEquation(AttitudeEquation, String)***.
  In practical terms, the user has to build the additional equation by extending the abstract class AttitudeEquation, call ***addAttitudeEquation(AttitudeEquation, String)***.

See [ORB_PRO_UseP mono numerical propagation] chapter for more details.

Add force model

For each state, a force model could be added to the list of forces used at each step by the propagator.

```
propagator.addForceModel(model, STATE1);
```

See [ORB_PHY_Home force model] chapter for more details.

Add event detector

The user could add :

- an EventDetector associated with a specific state using ***addEventDetector(EventDetector, String)***
- a MultiEventDetector associated with several states added to the propagator using ***addEventDetector(MultiEventDetector)***

See [MIS_EVT_Home events presentation] and [MIS_MEVEN_Home multi events presentation]

chapters for more details.

Add attitude provider

The user could add :

- an AttitudeProvider returning the independant attitude of one satellite using ***setAttitudeProvider(AttitudeProvider, String)***
- a MultiAttitudeProvider returning the attitude of one satellite correlated to the other using ***setAttitudeProvider(MultiAttitudeProvider, String)***

Note that method `getAttitudeProvider()` always returns a `MultiAttitudeProvider` whether you provided a single or multi attitude provider. If you provided a single `AttitudeProvider` then you can retrive it by calling `((MultiAttitudeProviderWrapper) multiNumericalPropagator.getAttitudeProvider()).getAttitudeProvider()`

Getting started with multi analytical propagation

Here is presented a basic instantiation of the multi analytical propagator.

Create propagator

The `MultiAnalyticalPropagator` only needs a date to be created:

```
final MultiAnalyticalPropagator propagator = new
MultiAnalyticalPropagator(initialDate);
```

This date permits to compute initial states and feed various internal maps with relevant data associated to known satellites IDs. At this step, no IDs are known by the propagator. There are two possibilities to add satellites with their ID:

- either use the other constructor that also handles a map of Propagators (each is mapped with the desired String ID). Each element of the map is put in propagator's map;
- or use the `addPropagator` method (see below).

Add propagators

Propagators mono can be added to the multi propagator using the method `addPropagator(String, Propagator)`. Each added state is defined by a unique ID. An error is raised if:

- the input spacecraft ID is empty or null.
- a propagator with the same ID was already added to the propagator.
- the mono propagator inherits from `NumericalPropagator` (`MultiAnalyticalPropagation` only propagates analytical propagators).

Important notes:

- Calling the constructor with a map in parameter calls `addPropagator(String, Propagator)` on each element of the map.
- Each mono propagator that is added has its event detectors list cleared, and its propagation mode

is reset to slave mode.
- However, the attitude providers that are hold by mono propagators are taken into account. Setters `setAttitudeProvider`, `setAttitudeProviderForces` and `setAttitudeProviderEvents` also exist to add a provider associated to a given satellite ID (and replace the existing one if there was one). Mechanisms of propagation of the attitude is copied from `AbstractPropagator`.
- The map of propagators can be retrieved with `getPropagators()` and the one of initial states with `getInitialStates()`.

Add event detector

The user could add:

- an `EventDetector` associated with a specific state using `addEventDetector(EventDetector, String)`;
- the method `addEventDetector(MultiEventDetector)` is however forbidden for the multi analytical propagator because event detection is performed in the same way as with the mono analytical propagation and the `MultiEventDetector` interface is not adapted to `AbstractPropagator` and `EventState` mechanisms. If a `MultiEventDetector` needs to be used, it shall also implement the `EventDetector` interface and be added thanks to `addEventDetector(EventDetector, String)`.

Hybrid propagation

Version 4.14 offers the possibility to perform hybrid multi propagation, that is to say perform numerical and analytical propagation at the same time.

New feature of the MultiNumericalPropagator

Contrary to the `MultiAnalyticalPropagator` that forbids the addition of `NumericalPropagators` to its propagation map, the `MultiNumericalPropagator` has a new method `addStateProvider(SpacecraftStateProvider, String)` that permits to add a `SpacecraftStateProvider` to a new class attribute of type `Map<String, SpacecraftStateProvider>`. The multi propagator works the same way than before for numerical propagation of states, but the class `MultiStateVectorInfo` was adapted to also compute states from the providers. These “analytical states” are not regarded when transforming spacecraft states to a state array vector though, they are only added virtually after numerical computation by calling their `getSpacecraftState()` method. Remarks:

- if no providers are added then propagator’s behavior is unchanged.
- the same conditions on providers than in MultiAnalyticalPropagator apply:
  - the ID shall not be known already and shall respect non nullity and emptiness
  - providers are set to slave mode if they are instances of Propagator, and their event detectors are cleared.
  - the input provider cannot be an instance of NumericalPropagator (numerical propagation shall be performed thanks to the standard propagation of SpacecraftStates).
- MultiNumericalPropagator shall contain at least one initial state used for numerical propagation, that is to say the hybrid mode cannot be used for analytical propagation only (use the MultiAnalyticalPropagator instead).

Contents

Interfaces

Interface	Summary	Javadoc
**MultiPropagator	This interface provides a way to propagate several states at any time.	...
**MultiAttitudeProvider	This interface is an attitude provider taking into account the state of the others satellites.	...
**MultiModeHandler	Common interface for all multi propagator mode handlers initialization.	...

Classes

Class	Summary	Javadoc
SpacecraftState	This class is the representation of a complete state holding orbit, attitude for forces and for events computation and additional states at a given date.	...
MultiNumericalPropagator	This class propagates several SpacecraftState using numerical integration.	...
MultiAnalyticalPropagator	This class propagates several SpacecraftState using analytical integration.	...
AbstractPropagator	This class propagates several SpacecraftState using analytical integration.	...
AttitudeEquation	This class represents attitude differential equations.	...
MultiIntegratedEphemeris	This class stores sequentially generated orbital parameters for later retrieval.	...
EventState	This class stores sequentially generated orbital parameters for later retrieval.	...
MultiPartialDerivativesEquations	This class computes the partial derivatives equations for one satellite. It is handled exactly as its mono-satellite counterpart PartialDerivativesEquations.	...

Catégorie :

- User Manual 4.14 Orbit Propagation

Menu de navigation

Outils personnels

Espaces de noms

- Page
- Discussion

Variantes

Affichages

- Lire
- Voir le texte source
- Historique
- Exporter en PDF

Plus

Rechercher

Rechercher Lire

PATRIUS

- Welcome

Evolutions

- Main differences between V4.14 and V4.13
- 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 4.14
- 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 4.14
- Tutorials 4.13.5
- Tutorials 4.12.1
- Tutorials 4.8.1
- 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 5 septembre 2024 à 15:31.

- Politique de confidentialité
- À propos de Wiki
- Avertissements

-