NumericalPropagationWithAttitudeSequence 4.4

NumericalPropagationWithAttitudeSequence 4.4

```java
public class NumericalPropagationWithAttitudeSequence {

    public static void main(String[] args) throws PatriusException,
IOException, URISyntaxException {

        // Patrius Dataset initialization (needed for example to get the UTC
time
        PatriusDataset.addResourcesFromPatriusDataset() ;

        // Recovery of the UTC time scale using a "factory" (not to duplicate
such unique object)
        final TimeScale TUC = TimeScalesFactory.getUTC();

        // Date of the orbit given in UTC time scale)
        final AbsoluteDate date = new AbsoluteDate("2010-01-01T12:00:00.000",
TUC);

        // Getting the frame with wich will defined the orbit parameters
        // As for time scale, we will use also a "factory".
        final Frame GCRF = FramesFactory.getGCRF();

        // Initial orbit
        final double sma = 7200.e+3;
        final double exc = 0.01;
        final double per = sma*(1.-exc);
        final double apo = sma*(1.+exc);
        final double inc = FastMath.toRadians(98.);
        final double pa = FastMath.toRadians(0.);
        final double raan = FastMath.toRadians(0.);
        final double anm = FastMath.toRadians(0.);
        final double MU = Constants.WGS84_EARTH_MU;

        final ApsisRadiusParameters par = new ApsisRadiusParameters(per, apo,
inc, pa, raan, anm, PositionAngle.MEAN, MU);
        final Orbit iniOrbit = new ApsisOrbit(par, GCRF, date);

        // We create a spacecratftstate
        final SpacecraftState iniState = new SpacecraftState(iniOrbit);

        // Initialization of the Runge Kutta integrator with a 2 s step
        final double pasRk = 2.;
        final FirstOrderIntegrator integrator = new
```

```java
ClassicalRungeKuttaIntegrator(pasRk);

        // Initialization of the propagator
        final NumericalPropagator propagator = new
NumericalPropagator(integrator);
        propagator.resetInitialState(iniState);

        // Forcing integration using cartesian equations
        propagator.setOrbitType(OrbitType.CARTESIAN);

//SPECIFIC
        // Adding attitude sequence
        final AttitudesSequence seqAtt = new AttitudesSequence();

        // Laws to be taken into account in the sequence
        final AttitudeLaw law1 = new ConstantAttitudeLaw(GCRF, new
Rotation(RotationOrder.ZYX, 0., 0., 0.));
        final AttitudeLaw law2 = new ConstantAttitudeLaw(GCRF, new
Rotation(RotationOrder.ZYX, FastMath.toRadians(45.), FastMath.toRadians(45.),
FastMath.toRadians(45.)));

        // Events that will switch from a law to another
        final double maxCheck = 10.;
        final double threshold = 1.e-3;
        final EventDetector event1 = new AOLDetector(0., PositionAngle.MEAN,
GCRF, maxCheck, threshold, Action.RESET_STATE);
        final EventDetector event2 = new
AOLDetector(FastMath.toRadians(180.), PositionAngle.MEAN, GCRF, maxCheck,
threshold, Action.RESET_STATE);

        //Adding switches
        seqAtt.addSwitchingCondition(law1, event1, true, false, law2);
        seqAtt.addSwitchingCondition(law2, event2, true, false, law1);

        propagator.setAttitudeProvider(seqAtt);
        seqAtt.registerSwitchEvents(propagator);
//SPECIFIC

        // Propagating 100s
        final double dt = 0.25*iniOrbit.getKeplerianPeriod();
        System.out.println(dt);
        final AbsoluteDate finalDate = date.shiftedBy(dt);
        final SpacecraftState finalState = propagator.propagate(finalDate);
        final Orbit finalOrbit = finalState.getOrbit();

        // Printing new date and true latitude argument
        System.out.println();
        System.out.println("Initial true latitude argument =
"+FastMath.toDegrees(iniOrbit.getLv())+" deg");
        System.out.println("New date = "+finalOrbit.getDate().toString(TUC)+"
deg");
```

```
        System.out.println("True latitude argument =
"+FastMath.toDegrees(finalOrbit.getLv())+" deg");
        // Printing attitude
        final double psi  =
finalState.getAttitude().getRotation().getAngles(RotationOrder.ZYX)[0];
        final double teta =
finalState.getAttitude().getRotation().getAngles(RotationOrder.ZYX)[1];
        final double phi  =
finalState.getAttitude().getRotation().getAngles(RotationOrder.ZYX)[2];
        System.out.println("Psi / GCRF  = "+FastMath.toDegrees(psi)+" deg");
        System.out.println("Teta / GCRF = "+FastMath.toDegrees(teta)+" deg");
        System.out.println("Phi / GCRF  = "+FastMath.toDegrees(phi)+" deg");

    }

}
```

Récupérée de
«
»

Menu de navigation

Rechercher

Rechercher Lire

PATRIUS

- [Welcome](#)

Evolutions

- [Main differences between V4.15 and V4.14](#)
- [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 4.15](#)
- [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](#)

Tutorials

Links

- [CNES freeware server](#)

Navigation

Outils

- Dernière modification de cette page le 3 octobre 2019 à 12:55.