User Manual 3.3 Projections

De Wiki
Aller à : navigation, rechercher
User Manual 3.3 Projections

Introduction

Scope

The scope of this section is to present the projections features available in Patrius library. Patrius provides classes to perform projections on an ellipsoid as well as various computation on the surface of an ellipsoid. Common available projections are:

- Mercator
- Flamsteed-Samson
- Identity projection

Javadoc

All the classes related to projections are in the following packages:

Library	Javadoc
Patrius	Package fr.cnes.sirius.patrius.projections

Links

More information about specific projections can be found using the follwing links:

- Mercator projection
- Flamsteed-Samson projection

Useful Documents

None as of now.

Package Overview

The projections package gathers:

- An interface for all projections: `IProjection`
- Some available projections : `Mercator`, `GeneralizedFlamsteedSamson` and `IdentityProjection`
- A ellipsoid with extended features: `ProjectionEllipsoid`

The projection package can be summarized with the following UML diagram:

Features Description

Projections provide three main features:

- Projection on an ellipsoid (and its inverse application)
- Discretisation between points on the surface of an ellipsoid
- Geometric computation on the surface of an ellipsoid using class ProjectionEllipsoid

Projection on an ellipsoid

Projection on an ellipsoid is a transformation taking geodetic coordinates (latitude, longitude, altitude) as input and returning 2D map coordinates (X, Y) as output. Inverse transformation takes 2D map coordinates (X, Y) as input and returns geodetic coordinates (latitude, longitude, altitude) as output.

Projection classes inherit projection interface `IProjection`. Available projections are:

- Mercator projection: `Mercator`. The Mercator projection is a cylindrical map projection which became the standard map projection for nautical purposes because of its ability to represent lines of constant course, known loxodromes, as straight segments. Mercator projection is conformal (angles are preserved) but not equivalent (areas are not preserved).

- Flamsteed-Samson projection: `GeneralizedFlamsteedSamson`. This projection is also known as the sinusoidal projection. The sinusoidal projection is equal-area and preserves distances along the horizontals but is not conformal (angles are not preserved).

- Identity projection: `IdentityProjection`. Identity projection is the projection [latitude, longitude, altitude] => [X = latitude, Y = longitude].

All projections provide:

- direct transformation using method `applyTo()`. It returns 2D map coordinates (X, Y) from geodetic coordinates (latitude, longitude, altitude).
- inverse transformation using method `applyInverseTo()`. It returns geodetic coordinates (latitude, longitude, altitude) from 2D map coordinates (X, Y).

Discretization

All projection classes provide various discretization methods between geodetic points.

- Discretization between two projected points using the method `discretize()`. Maximum distance between discretized point has to be provided.
- Discretization along a polygon line of geodetic coordinates using the method `discretizeAndApplyTo()`. Maximum distance between discretized point has to be provided as well as the discretization strategy.

The discretization strategy is provided with the enumeration `EnumLineProperty`. It offers the following possibilities:

- `STRAIGHT`: straight line between two geodetic points.

- `GREAT_CIRCLE`: arc being the shortest way to connect two geodetic points on an ellipsoid. The center of the ellipsis is the ellipsoid center.
- `STRAIGHT_RHUMB_LINE`: arc between two geodetic points crossing all meridians of longitude at the same angle. It corresponds to a path of constant bearing as measured relative to true north. On Mercator projection, it is represented by a straight line.

Other methods are available. For more information, refer to the javadoc of AbstractProjection class.

Computation on the surface of an ellipsoid

A specific ellipsoid class has been defined to handle projections: `ProjectionEllipsoid`. This class inherits the class `ExtendedOneAxisEllipsoid`. It provides many useful projection-related computations on the surface of an ellipsoid. Available features of `ProjectionEllipsoid` are:

- Distance computation:

Orthodromic distance using method `computeOrthodromicDistance()`. Orthodromic distance is the shortest path between two points. Loxodromic distance using method `computeLoxodromicDistance()`. Loxodromic distance follows path of constant bearing: this is a straight line on Mercator projection. Meridional distance using method `computeMeridionalDistance()`. Meridional distance is the shortest distance from one point to the equator (along a meridian).

On next image, Loxodromic distance is in red, orthodromic distance in blue:

- Azimuth computation:

Bearing using method `computeBearing()`. Spherical azimuth using method `computeSphericalAzimuth()`.

- Other computations:

Point along loxodrome, given a point, a distance from this point and an azimuth from this point, using method `computePointAlongLoxodrome()`. Point along orthodrome, given a point, a distance from this point and an azimuth from this point, using method `computePointAlongOrthodrome()`.

Other methods are available. For more information, refer to the javadoc of ProjectionEllipsoid class.

Getting Started

Projections provide three main features:

- Projection on an ellipsoid (and its inverse application)
- Discretisation between points on the surface of an ellipsoid
- Geometric computation on the surface of an ellipsoid using class `ProjectionEllipsoid`

Projection on an ellipsoid

First an ellipsoid and a projection have to be defined, here using a simple centered Mercator projection:

```
final ProjectionEllipsoid ellipsoid = new ProjectionEllipsoid(6378137.0, 1. /
298.257223563, null, "Earth");
final Mercator projection = new Mercator(0., ellipsoid);
```

Then the created projection can be used in different ways:

- Projection of geodetic coordinates:

```
final GeodeticPoint toulouse = new GeodeticPoint(0.758011794744558,
0.0261405281982074, 256.);
final Vector2D projectedPoint = projection.applyTo(toulouse);
```

- Retrieve geodetic coordinates from Mercator 2D coordinates:

```
final GeodeticPoint geodeticCoordinates =
projection.applyInverseTo(projectedPoint.getX(), projectedPoint.getY());
```

Discretization

First an ellipsoid and a projection have to be defined, here using a simple centered Mercator projection:

```
final ProjectionEllipsoid ellipsoid = new ProjectionEllipsoid(6378137.0, 1. /
298.257223563, null, "Earth");
final Mercator projection = new Mercator(0., ellipsoid);
```

Then the created projection can be used in different ways:

- Discretization between two projected points with a maximum distance between points of 1km:

```
final GeodeticPoint toulouse = new GeodeticPoint(0.758011794, 0.026140528,
256.);
final GeodeticPoint london = new GeodeticPoint(0.898844565, 0.002268928,
25.);
final Vector2D projectedToulouse = projection.applyTo(toulouse);
final Vector2D projectedLondon = projection.applyTo(london);
final List<Vector2D> points = projection.discretize(projectedToulouse,
projectedLondon, 1000., true);
```

- Rhumb discretization along a polygon line with a maximum distance between points of 1km followed by projection:

```
final GeodeticPoint toulouse = new GeodeticPoint(0.758011794, 0.026140528,
256.);
final GeodeticPoint london = new GeodeticPoint(0.898844565, 0.002268928,
25.);
final List<GeodeticPoint> list = new ArrayList<GeodeticPoint>();
list.add(toulouse);
```

```
list.add(london);
final List<Vector2D> points = projection.discretizeAndApplyTo(list ,
EnumLineProperty.STRAIGHT_RHUMB_LINE, 1000.);
```

- etc. Other similar discretization features are available. See javadoc for more information.

Computation on the surface of an ellipsoid

First an ellipsoid has to be defined:

```
final ProjectionEllipsoid ellipsoid = new ProjectionEllipsoid(6378137.0, 1. /
298.257223563, null, "Earth");
final GeodeticPoint toulouse = new GeodeticPoint(0.758011794, 0.026140528,
256.);
final GeodeticPoint london = new GeodeticPoint(0.898844565, 0.002268928,
25.);
```

Then this ellipsoid can be used to:

- Compute distances on the surface of the ellipsoid:

```
final double orthodromicDistance =
ellipsoid.computeOrthodromicDistance(toulouse, london);
final double loxodromicDistance =
ellipsoid.computeLoxodromicDistance(toulouse, london);
final double meridionalDistance =
ellipsoid.computeMeridionalDistance(toulouse.getLatitude());
```

- Compute azimuth angles on the surface of the ellipsoid:

```
final double bearing = ellipsoid.computeBearing(toulouse, london);
final double sphericalAzimuth = ellipsoid.computeSphericalAzimuth(toulouse,
london);
```

- etc. Other similar features are available. See javadoc for more information.

Contents

Interfaces

Interface	Summary	Javadoc
IProjection	Interface for projections on a 3D body.	...

Classes

Class	Summary	Javadoc

EnumLineProperty	Enumeration of points connecting strategies on an ellipsoid.	...
IdentityProjection	Identity projection.	...
GeneralizedFlamsteedSamson	Flamsteed-Samson projection.	...
Mercator	Mercator projection.	...
ProjectionEllipsoid	Ellipsoid with extended features (features related to projections).	...

Tutorials

Tutorial 1

Modèle:SpecialInclusion prefix=$theme sub section="Tuto1"/

Tutorial 2

Modèle:SpecialInclusion prefix=$theme sub section="Tuto2"/

Tips & Tricks

Modèle:SpecialInclusion prefix=$theme sub section="Tips"/

Menu de navigation

Outils personnels

Espaces de noms

Variantes

Affichages

- Historique
- Exporter en PDF

Plus

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
- User Manual 3.4.1
- User Manual 3.3

Tutorials

- Tutorials 4.15
- 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

Outils

-