Package 'asteRisk'

May 7, 2026

Type Package

Title Computation of Satellite Position

Version 1.4.5

Description Provides basic functionalities to calculate the position of satellites given a known state vector. The package includes implementations of the SGP4 and SDP4 simplified perturbation models to propagate orbital state vectors, as well as utilities to read TLE files and convert coordinates between different frames of reference. Several of the functionalities of the package (including the high-precision numerical orbit propagator) require the coefficients and data included in the 'asteRiskData' package, available in a 'drat' repository. To install this data package, run 'install.packages(``asteRiskData", repos=``https://rafael-ayala.github.io/drat/")'.
Felix R. Hoots, Ronald L. Roehrich and T.S. Kelso (1988) <https://celestrak.org/NORAD/documentation/spacetrk.pdf>.
David Vallado, Paul Crawford, Richard Hujsak and T.S. Kelso (2012) <doi:10.2514/6.2006-6753>.
Felix R. Hoots, Paul W. Schumacher Jr. and Robert A. Glover (2014) <doi:10.2514/1.9161>.

Acknowledgements The development of this work is supported by the following grants: a KAKENHI Grant-in-Aid for Research Activity Start-up Grant Number 21K20645 to Rafael Ayala, a JSPS Postdoctoral Fellowship for Research in Japan (Standard) Grant Number P20810 to Lara Sellés Vidal (Overseas researcher under Postdoctoral Fellowship of Japan Society for the Promotion of Science), and grants by the Spanish Ministry of Science and Innovation (grant code PID2019-105471RB-I00) and the Regional Government of Andalusia (grant code P18-RT-1060) to David Ruiz.

License GPL-3

Depends R (>= 4.0)

LinkingTo Rcpp, RcppParallel, BH, RcppEigen

SystemRequirements GNU make

Imports deSolve, nanotime, stats, onion, Rcpp, RcppParallel, utils, gsl, polynom, tools, httr

Suggests asteRiskData, knitr, formatR, webshot, BiocStyle, RUnit, plotly, lazyeval, dplyr, ggmap, rmarkdown, markdown

BugReports https://github.com/Rafael-Ayala/asteRisk/issues

Additional_repositories https://rafael-ayala.github.io/drat/

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation yes

Maintainer Rafael Ayala <rafaelayalahernandez@gmail.com>

Author Rafael Ayala [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9332-4623>),
Daniel Ayala [aut] (ORCID: <https://orcid.org/0000-0003-2095-1009>),
David Ruiz [aut] (ORCID: <https://orcid.org/0000-0003-4460-5493>),
Pablo Hernandez [aut] (ORCID: <https://orcid.org/0009-0000-9279-6744>),
Lara Selles Vidal [aut] (ORCID:
<https://orcid.org/0000-0003-2537-6824>)

Date/Publication 2025-11-02 06:10:50 UTC

Contents

`BCItoKOE` *Calculate Keplerian orbital elements from BCI coordinates*

Description

Keplerian orbital elements are a set of six parameters used to described the orbits of celestial objects, including satellites. While satellites do not follow a perfectly Keplerian orbit, their state at any point can be defined by the orbital parameters that they would have if they were located at the same position with the same velocity following a perfectly Keplerian orbit (i.e., if perturbations were absent). These are called osculating orbital elements.

Keplerian orbital elements can be unequivocally determined from a satellite if its position and velocity are known. This function calculates orbital elements from the position and velocity of a satellite in an BCI (body-centered inertial) frame of reference. The elements (such as the equatorial plane) with respect to which the resulting orbital elements will be defined are the same as those used for the BCI frame of reference. The function calculates the six standard orbital elements, plus some alternative elements useful for the characterization of special orbits, such as circular ones or orbits with no inclination.

By default, the function assumes the central body is the Earth and uses the GM of the Earth in the TDB frame. This can be changed by providing the GM of the desired central body in m^3/s^2.

Usage

```
BCItoKOE(position_BCI, velocity_BCI, centralBodyGM=GM_Earth_TDB)
```

Arguments

`position_BCI`	Vector with the X, Y and Z components of the position of an object in a BCI frame, in m.
`velocity_BCI`	Vector with the X, Y and Z components of the velocity of an object in a BCI frame, in m/s.
`centralBodyGM`	GM of the central body in m^3/s^2. By default, the GM of the Earth in the TDB frame is used.

Value

A list with the following standard and alternative orbital elements:

`semiMajorAxis` Semi-major axis of orbital ellipse in meters.

`eccentricity` Numerical eccentricity of the orbit. Eccentricity measures how much the orbit deviates from being circular.

`inclination` Inclination of the orbital plane in radians. Inclination is the angle between the orbital plane and the equator.

`meanAnomaly` Mean anomaly of the orbit in radians. Mean anomaly indicates where the satellite is along its orbital path, and is defined as the angle between the direction of the perigee and the hypothetical point where the object would be if it was moving in a circular orbit with the same period as its true orbit after the same amount of time since it last crossed the perigee had ellapsed.

`argumentPerigee` Argument of perigee in radians. This is the angle between the direction of the ascending node and the direction of the perigee (the point of the orbit at which the object is closest to the central body).

`longitudeAscendingNode` Longitude of the ascending node (also called right ascension of the ascending node) in radians. This is the angle between the direction of the ascending node (the point where the satellite crosses the equatorial plane moving north) and the direction of the First Point of Aries (which indicates the location of the vernal equinox).

`trueAnomaly` True anomaly of the orbit in radians. Unlike mean anomaly, true anomaly is the angle between the direction of the perigee and the actual position of the satellite.

`argumentLatitude` Argument of latitude of the orbit in radians. Defined as the angle between the equator and the position of the satellite. It is useful to define the position of satellites in circular orbits, where the argument of perigee and true anomaly are not well defined.

`longitudePerigee` Longitude of perigee of the orbit in radians. Defined as the angle between the vernal equinox and the perigee. It is useful for cases of orbits with 0 inclination, where the longitude of the ascending node and the argument of perigee are not well defined.

`trueLongitude` Longitude of perigee of the orbit in radians. Defined as the angle between the vernal equinox and the position of the satellite. It is useful for cases of circular orbits with 0 inclination, where the longitude of the ascending node, the argument of perigee and true anomaly are not well defined.

References

https://www.gsc-europa.eu/system-service-status/orbital-and-technical-parameters https://celestrak.org/columns/v02n01/
https://www.faa.gov/about/office_org/headquarters_offices/avs/offices/aam/cami/library/online_libraries/aerospace_medicin

Examples

```
# The following were the position and velocity of satellite MOLNIYA 1-83
# the 25th of June, 2006 at 00:33:43 UTC in the GCRF frame (in m and m/s).

position_GCRF <- c(-14471729.582, -4677558.558, 9369.461)
velocity_GCRF <- c(-3251.691, -3276.008, 4009.228)

# Let's calculate the orbital elements of the satellite at that time

orbital_elements <- BCItoKOE(position_GCRF, velocity_GCRF)
```

---

calculateRazel *Calculates azimuth, elevation and range of a given object*

Description

The horizontal coordinate system, also called azimuth-elevation system, uses the local horizon of an observer as its fundamental plane. In it, a given point is defined by 2 main angles: azimuth and elevation. Azimuth defines the angle of the point around the horizon in the X-Y plane, measured from the true North and usually increasing towards the East. Elevation is the angle between the object and the X-Y plane. Finally, the range defines the distance between the observer and the point.

This function calculates the azimuth, elevation and range given the coordinates of an observed satellite and of an observer. Both sets of coordinates must be provided as Cartesian geocentric coordinates in ITRF.

Usage

```
calculateRazel(geocentricObserver, geocentricSatellite, degreesOutput=TRUE)
```

Arguments

`geocentricObserver`	Vector with the X, Y and Z components of the position of the observer in ITRF frame.
`geocentricSatellite`	Vector with the X, Y and Z components of the position of the satellite in ITRF frame.
`degreesOutput`	Logical indicating if the output should be in sexagesimal degrees. If `degreesOutput=FALSE`, the output will be in radians.

Value

A vector with three elements, corresponding to the azimuth and elevation in degrees (or radians if specified) and the range in the same unit as the provided Cartesian coordinates.

References

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Examples

```
# The following were the coordinates of Italsat-2 in ITRF the 27th of June, 2006
# at 00:58:29.34 UTC, in meters.

italsat_ITRF <- c(-37325542.8, 19152438.3, 138384.5)

# Let us calculate its azimuth, elevation and range for an observer from Tokyo.
# The latitude and longitude of the city are 35.6762 degrees North, 139.6503
# degrees East. Let's assume an observer placed at sea level (0 m)
# We first convert these coordinates to ITRF:

observer_ITRF <- LATLONtoITRF(c(35.6762, 139.6503, 0), degreesInput=TRUE)

# We can now calculate the azimuth and elevation:

razel <- calculateRazel(observer_ITRF, italsat_ITRF, degreesOutput=TRUE)
razel[1] # Azimuth
razel[2] # Elevation
```

`dateTimeToMJD` *Calculate Modified Julian Date for a given date and time*

Description

The Julian Date (JD) of a given date and time is the number of days since noon of Monday 1st of January 4713 BC, including a fractional part. Modified Julian Date (MJD) are instead the number of days since 00:00 of November 17th, 1858. The difference JD and MJD for a given instant is always 2400000.5, which is the JD of the reference time for MJD.

This function calculates the MJD of a date and time, provided as a date-time character string in UTC time. The output refers by default to the MJD in UTC, but different time systems can be chosen: UTC (Coordinated Universal Time), UT1 (Universal Time), TT (Terrestrial Time) and TDB (Barycentric Dynamical Time).

Usage

```
dateTimeToMJD(dateTime, timeSystem="UTC")
```

Arguments

`dateTime`	Date-time string with the date and time in UTC corresponding to the provided geodetic coordinates.
`timeSystem`	Time system into which the MJD should be calculated. Should be one from "UTC" (Coordinated Universal Time; default), "UT1" (Universal Time), "TT" (Terrestrial Time) and "TDB" (Barycentric Dynamical Time).

Value

The MJD for the specified date and time in the chosen time system.

References

https://gssc.esa.int/navipedia/index.php/Julian_Date https://gssc.esa.int/navipedia/index.php/Transformations_between_Tim

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# Let's calculate the MJD of the 12th of June, 2000 at 10:00:00 UTC time, in UTC

MJD_UTC <- dateTimeToMJD("2000-06-12 10:00:00", timeSystem = "UTC")

# Let's now calculate the MJD for the same instant in TDB:

MJD_TDB <- dateTimeToMJD("2000-06-12 10:00:00", timeSystem = "TDB")

# We can now calculate the difference in seconds, which matches the difference
# between UTC and TDB for that day:

(MJD_UTC - MJD_TDB) * 86400
}
```

---

deg2rad *Converts an angle in degrees to radians*

Description

This function converts an angle in degrees to radians.

Usage

```
deg2rad(degrees)
```

Arguments

degrees	Value of the angle in degrees.

Value

The corresponding value of the angle in radians.

Examples

```
deg2rad(180)
```

`evaluateSPKSegment` *Evaluate a given SPK segment*

Description

SPK (Spacecraft and Planet Kernel) is a binary file format developed by NAIF to store ephemerides (trajectory) of celestial bodies and spacecraft in the Solar System. A detailed description of the SPK file format can be found in NAIF's documentation (https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html).

Each SPK file contains several segments, with each segment comprising a summary (or descriptor), a name and an array of double precision elements. Each segment is conceptually equivalent to an array in the context of generic DAF files. There are several types of SPK segments defined by NAIF, each identified by an SPK type code currently ranging from 1 to 21 (some intermediate values are not used or not available for general public use). Each segment type provides ephemerides information in a different way. Note that the segments stored in a single SPK file can be of different types. A detailed description of the organization of the arrays for each SPK type can be found at https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html

SPK files can be read with readBinSPK. This function allows the evaluation of one segment of an SPK file to any target times. The evaluated segment should be one of the elements in the list of segments returned by calling readBinSPK on an SPK file.

Evaluation at target times of the SPK segment produces state vectors comprising X, Y and Z components for position and velocity for the target body. The target body, center body and frame of reference are those indicated in the corresponding summary of the SPK segment.

The specific algorithm through which an SPK segment is evaluated depends on the type of SPK segment. For a detailed description, see the documentation of readBinSPK or NAIF's documentation.

The output values for all types of segments have been verified to match those of CSPICE to a precision of `sqrt(.Machine$double.eps)`.

Note that, in addition to position and velocity components, `evaluateSPK` will also produce X, Y and Z components of acceleration for segments of types 1, 2, 3, 8, 9, 12, 13, 14, 18, 19, 20 and 21. This is possible because all of these segments provide some sort of interpolation polynomial coefficients, which can be differentiated to obtain acceleration values. However, note that said coefficients are, in principle, not intended to be used to calculate acceleration values, and SPICE does not return these. Therefore, proceed with caution if using acceleration values obtained in this way. In the specific case of SPK segments of types 1 and 21, the provided coefficents are for an interpolation polynomial for acceleration, and therefore are probably the most reliable. In particular, for target times exactly matching the reference epoch of any of the data points included in the segment, the acceleration value should match the original value used to fit the interpolation polynomial.

Usage

```
evaluateSPKSegment(segment, targetEpochs)
```

Arguments

segment	Single SPK segment to be evaluated. The segment should have the same structure as that of segments returned by the readBinSPK function. Note said function returns all segments contained in the read SPK file; only one of those should be selected and provided here.
targetEpochs	Numeric vector indicating the target epochs at which the segment should be evaluated. The epochs should be provided in TDB seconds since J2000, also known as ephemeris time in SPICE. Note that all target epochs should be larger than the start epoch of the segment, and smaller than the end epoch of the segment, both of which are specified in the corresponding segment summary. Segments of type 1 and 21 are an exception. There are segments of these types where the final record covers epochs up to a time larger than the global end epoch of the corresponding segment; in cases where epochs larger than the global end epoch of the segment, but smaller than the end epoch of the last data record, these epochs will also be evaluated.

Value

A matrix with 7 (for SPK segments of types 5, 10 and 15) or 10 (for all other types) columns. Each row in the matrix represents a target time of evaluation. Column 1 provides the epochs of evaluation. Columns 2-4 provide position components. Columns 5-7 provide velocity components. Column 8-10, when present, provide acceleration components.

References

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif_ids.h
https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/frames.html Shampine, L. F. and Gordon, M. K., Computer Solution of Ordinary Differential Equations: The Initial Value Problem, 1975 Robert Werner, SPICE spke01 math, 2022. https://doi.org/10.5270/esa-tyidsbu

Examples

```
# The file vgr2_jup230.bsp provided with the package includes information for the
# Jupiter flyby of Voyager 2

testSPK <- readBinSPK(paste0(path.package("asteRisk"), "/vgr2_jup230.bsp"))
length(testSPK$segments)

# It contains a single segment.

testSegment <- testSPK$segments[[1]]

# Check the initial and end epochs of the interval covered by the segment

testSegment$segmentSummary$initialEpoch
testSegment$segmentSummary$finalEpoch

# Evaluate at target epochs

evaluateSPKSegment(testSegment, c(-649364400, -649364374.68,  -647364600, -645364800))
```

`GCRFtoITRF` *Convert coordinates from GCRF to ITRF*

Description

The GCRF (Geocentric Celestial Reference Frame) frame of reference is an Earth-centered inertial coordinate frame, where the origin is placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The X-axis is aligned with the mean equinox of Earth at 12:00 Terrestrial Time on the 1st of January, 2000, and the Z-axis is aligned with the Earth´s rotation axis.

It is almost equivalent to the J2000 frame of reference (also called EME2000), and in some contexts it is also referred to as ICRF frame (although in its strict definition, the origin of coordinates is placed at the barycenter of the Solar System).

In the ITRF (International Terrestrial Reference Frame), the origin is also placed at the center of mass of Earth, but the frame rotates with respect to the stars to remain fixed with respect to the Earth surface as it rotates. The Z-axis extends along the true North as defined by the IERS reference pole, and the X-axis extends towards the intersection between the equator and the Greenwich meridian at any time.

The coordinates and velocities input and calculated with the high-precision orbital propagator (hpop) are in the GCRF frame of reference.

This function requires the `asteRiskData` package, which can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
GCRFtoITRF(position_GCRF, velocity_GCRF, dateTime)
```

Arguments

`position_GCRF`	Vector with the X, Y and Z components of the position of an object in GCRF frame, in m.
`velocity_GCRF`	Vector with the X, Y and Z components of the velocity of an object in GCRF frame, in m/s.
`dateTime`	Date-time string with the date and time in UTC corresponding to the provided position and velocity vectors. This specifies the time for which the conversion from GCRF to ITRF coordinates will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of GCRF coordinates refers varies with time due to the motion of Earth.

Value

A list with two elements representing the position and velocity of the satellite in the ITRF (International Terrestrial Reference Frame) frame of reference. Position values are in m, and velocity values are in m/s. Each of the two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order.

References

https://celestrak.org/columns/v02n01/

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following were the position and velocity of satellite MOLNIYA 1-83
# the 25th of June, 2006 at 00:33:43 UTC in the GCRF frame (in m and m/s).

position_GCRF <- c(-14471729.582, -4677558.558, 9369.461)
velocity_GCRF <- c(-3251.691, -3276.008, 4009.228)

# Let´s convert them into the ITRF frame

coordinates_ITRF <- GCRFtoITRF(position_GCRF, velocity_GCRF, "2006-06-27 00:58:29.34")
}
```

GCRFtoLATLON *Convert coordinates from GCRF to geodetic latitude, longitude and altitude*

Description

The GCRF (Geocentric Celestial Reference Frame) frame of reference is an Earth-centered inertial coordinate frame, where the origin is placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The X-axis is aligned with the mean equinox of Earth at 12:00 Terrestrial Time on the 1st of January, 2000, and the Z-axis is aligned with the Earth´s rotation axis. This function converts position in GCRF to geodetic latitude, longitude and altitude, which can be considered to be a non-inertial, Earth-centered frame of reference.

This function requires the `asteRiskData` package, which can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
GCRFtoLATLON(position_GCRF, dateTime, degreesOutput=TRUE)
```

Arguments

`position_GCRF`	Vector with the X, Y and Z components of the position of an object in TEME frame, in m.
`dateTime`	Date-time string with the date and time in UTC corresponding to the provided position vector. This specifies the time for which the conversion from GCRF to geodetic coordinates will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of GCRF coordinates refers varies with time due to the motion of Earth.
`degreesOutput`	Logical indicating if the output should be in sexagesimal degrees. If `degreesOutput=FALSE`, the output will be in radians.

Value

A vector with three elements, corresponding to the latitude and longitude in degrees (or radians if specified) and the altitude in m.

References

https://arc.aiaa.org/doi/10.2514/6.2006-6753

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Let´s calculate the position and velocity of the satellite 1 day later

state_1day_TEME <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                         Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)

# We can now convert the results in TEME frame to GCRF frame, previously
# multiplying by 1000 to convert the km output of sgdp4 to m

state_1day_GCRF <- TEMEtoGCRF(state_1day_TEME$position*1000,
                              state_1day_TEME$velocity*1000,
                              "2006-06-27 00:58:29.34")

# Finally, we convert the results in GCRF frame to geodetic latitude, longitude
# and altitude

state_1day_geodetic <- GCRFtoLATLON(state_1day_GCRF$position, "2006-06-27 00:58:29.34")
}
```

---

getLatestSpaceData *Retrieves the latest space data*

Description

The asteRiskData package provides the data and coefficients required for calculation of forces for hpop and other functions such certain conversions between reference frames. Some of the data tables included in the package are updated periodically with new data. These include Earth

orientation parameters, space weather data and solar and geomagnetic storms. In order to perform the calculations dependent on such data for the most recent times, the latest available data must be retrieved.

This function automatically updates the data tables, enabling such calculations for the most recent times.

Usage

```
getLatestSpaceData(targets="all")
```

Arguments

targets	Character vector specifying the data that should be updated. It should be a vector containing one or more of the following strings: "all" (to update all data), "EOP" (Earth orientation parameters), "SW" (space weather), "SS" (solar storms) or "GS" (geomagnetic storms). By default, all data are updated.

Value

This function is invoked for its side effect, which is updating the data tables used internally for calculations requiring `asteRiskData` package, such as those performed by hpop.

References

http://www.celestrak.org/SpaceData/EOP-All.txt https://celestrak.org/SpaceData/SW-All.txt https://sol.spacenvironment.net/

Examples

```
if(interactive()) {
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The table of Earth orientation parameters distributed with asteRiskData
# comprises data up to the 21st of March, 2021

asteRiskData::earthPositions[nrow(asteRiskData::earthPositions),]

# The table can be easily updated to include the most recent available data

getLatestSpaceData(targets="all")
asteRiskData::earthPositions[nrow(asteRiskData::earthPositions),]
}
}
```

hpop *High-precision numerical orbital propagator*

Description

Given the position and velocity of a satellite at a given time (in the ICRF system of coordinates centered on the Solar System Barycenter, any of the main planets, Earth's Moon or Pluto), propagates its position by calculating its acceleration (based on a force model) and solving the resulting second-order ODE through numerical integration. This allows propagation of orbits with considerably higher accuracy than other propagators such as SGP4 and SDP4, but at the expense of a much higher computational cost. The forces and effects currently considered are gravitational attraction by the Earth (using the GGM05C gravity model, with spherical harmonics up to degree and order of 360); effects of Earth ocean and solid tides; gravitational attraction by the Moon (using the GRGM1200B gravity model with spherical harmonics up to degree and order of 1200), effects of solid Moon tides (currently using an ellastic Moon model), Sun and planets (considered as point masses); solar radiation pressure; atmospheric drag, and relativistic effects. The force field is based on the forces described in Satellite Orbits: Models, Methods and Applications (Oliver Montenbruck and Eberhard Gill) and Fundamentals of Astrodynamics and Applications (David Vallado). The NRLMSISE-00 model is used to calculate atmospheric density for the calculation of atmospheric drag. The FES2014 model is used to calculate Earth geopotential model corrections due to ocean tides. As mentioned before, the central body for the frame of reference can be any of the Solar System Barycenter (SSB), any of the main planets, Earth's Moon or Pluto. By default, it is assumed to be Earth, corresponding to GCRF (Geocentric ICRF). The initial position will be checked against the position of said celestial bodies, to identify if it falls under the Laplacian gravitational sphere of influence of any of them. If this is the case, and it differs from the specified central body, the coordinate system will be changed to be centered on the celestial body whose sphere of influence includes the object of interest. This avoids instability in propagation. The high-precision numerical orbital propagator requires the `asteRiskData` package, which provides the data and coefficients required for calculation of the modeled forces. `asteRiskData` can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
hpop(position, velocity, dateTime, times, satelliteMass, dragArea,
     radiationArea, dragCoefficient, radiationCoefficient,
     earthSphericalHarmonicsDegree=130, solidEarthTides=TRUE,
     oceanTides=TRUE, moonSphericalHarmonicsDegree=150, solidMoonTides=TRUE,
     centralBody="Earth", autoCentralBodyChange=TRUE, ...)
```

Arguments

`position`	Initial position of the satellite in the GCRF system of coordinates. Should be provided as a numeric vector with 3 components that indicate the X, Y and Z components of the position in meters.
`velocity`	Initial velocity of the satellite in the GCRF system of coordinates. Should be provided as a numeric vector with 3 components that indicate the X, Y and Z components of the position in meters/second.

`dateTime` Date time string in the YYYY-MM-DD HH:MM:SS format indicating the time corresponding to the initial position and velocity, in UTC time.

`times` Vector with the times at which the position and velocity of the satellite should be calculated, in seconds since the initial time.

`satelliteMass` Mass of the satellite in kilograms.

`dragArea` Effective area of the satellite for atmospheric drag in squared meters. If the way that a satellite will orient with respect to its velocity is not known, a mean cross-sectional area should be calculated assuming that the orientation of the satellite with respect to its velocity will vary uniformly. A decent estimate can be obtained with a flat-plate model, where the satellite is considered to be parallelepiped-shaped. The mean effective area can then be calculated as CSA = (S1 + S2 + S3 (+S4))/2, where S1, S2 and S3 are the areas of the three perpendicular surfaces of the model and S4 is an optional term to account for the area of solar panels (potential masking between the solar panels and the main surfaces is not considered; this might be partially accounted for by introducing a factor to reduce the calculated effective area).

`radiationArea` Effective area of the satellite subject to the effect of radiation pressure in squared meters.

`dragCoefficient`
Drag coefficient (Cd) used for the calculation of atmospheric drag. For low Earth-orbiting satellites, a value of 2.2 is frequently employed if a better approximation is not available.

`radiationCoefficient`
Coefficient for the force resulting from radiation pressure. This parameter is usually referred to as reflectivity coefficient (Cr) and the value varies for different satellites and orbits. If unknown, a value of 1.2 is usually a decent approximation.

`earthSphericalHarmonicsDegree`
Maximum degree and order that should be considered when calculating the Earth geopotential model. The model will be complete up to the specified degree/order, i.e., all zonal, sectorial and tesseral spherical harmonics will be calculated. The maximum possible value is 360, since that is the highest degree and order of the Stokes' coefficients provided in the GGM05C model. Note that spherical harmonics for Earth gravity field will only be used if Earth is the central body for propagation; otherwise, only a point-mass attraction will be calculated.

`solidEarthTides`
Logical indicating if corrections of the Cnm and Snm Stokes' coefficients for the geopotential model due to solid Earth tides should be performed, following IERS 2010 procedures and considering anelasticity of the Earth.

`oceanTides` Logical indicating if corrections of the Cnm and Snm Stokes' coefficients for the geopotential model due to ocean tides should be performed, using the FES2014 oceanic tides model.

`moonSphericalHarmonicsDegree`
Maximum degree and order that should be considered when calculating the Moon gravity model. The model will be complete up to the specified degree/order, i.e., all zonal, sectorial and tesseral spherical harmonics will be calculated. The

maximum possible value is 1200, since that is the highest degree and order of the Stokes' coefficients provided in the GRGM1200B model. Note that spherical harmonics for Moon gravity field will only be used if Moon is the central body for propagation; otherwise, only a point-mass attraction will be calculated.

`solidMoonTides` Logical indicating if corrections of the Cnm and Snm Stokes' coefficients for the lunar gravity model due to solid Moon tides should be performed, following the procedure described by William and Boggs, 2015 using an elastic Moon model. Corrections are applied to the C20, C21, C22, S21 and S22 coefficients.

`centralBody` Character string indicating the celestial body on which the supplied initial position (in ICRF) are centered. Should be one of "SSB" (meaning Solar System Barycenter), "Mercury", "Venus", "Earth", "Moon", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune" or "Pluto". The initial position will be checked against the position of said celestial bodies, to identify if it falls under the Laplacian gravitational sphere of influence of any of them. If this is the case, and it differs from the specified central body, the coordinate system will be changed to be centered on the celestial body whose sphere of influence includes the object of interest.

`autoCentralBodyChange` Logical indicating if the celestial object used as the center of coordinates should be automatically updated during propagation based on the radii of the spheres of influence of the main planets, the Moon and Pluto. By default, `autoCentralBodyChange=TRUE`.

`...` Additional parameters to be passed to ode to control how numerical integration is performed. By default, the RADAU5 solver is used.

Value

A data frame with the results of the numerical integration at the requested times. Each row contains the results for one of the requested times. The data frame contains 11 columns: time (indicating the time for the corresponding row in seconds since the initial time), positionX, positionY, positionZ (indicating the X, Y and Z components of the position for that time in meters), velocityX, velocityY and velocityZ (indicating the X, Y and Z components of the velocity for that time in meters/second), accelerationX, accelerationY, accelerationZ (indicating the X, Y and Z components of the acceleration for that time in meters/second^2) and centralBody, indicating the central body of the frame of reference for the results for the corresponding time. Positions and velocities are returned in the ICRF frame of reference, centered in the celestial body specified in column centralBody. If `autoCentralBodyChange=TRUE`, the celestial body whose sphere of influence includes the object of interest will be automatically used as the central body. Additionally, if transitions in or out of the spheres of influence of the main celestial bodies are detected during propagation of the trajectory, the central body will be automatically modified accordingly. If `autoCentralBodyChange=FALSE`, such automatic changes of the central body will not be performed, and instead the user-specified central body will be used at all times. Note, however, that it is not recommended to perform propagation in a frame center at an object different than the celestial body whose sphere of influence includes the target of propagation, since this can lead to a substantial loss of accuracy. For details, see M. Vautier, 2008. Note that, if none of the spheres of influence of the planets, Moon or Pluto included the object of interest, the center of the ICRF frame will be placed at the Solar System Barycenter.

References

Satellite Orbits: Models, Methods and Applications. Oliver Montenbruck and Eberhard Gill. Fundamentals of Astrodynamics and Applications. David Vallado. https://www.mathworks.com/matlabcentral/fileexchange/5516 high-precision-orbit-propagator https://ccmc.gsfc.nasa.gov/modelweb/models/nrlmsise00.php https://digitalcommons.usu.ed https://iopscience.iop.org/article/10.1088/1742-6596/911/1/012009/pdf https://www.sciencedirect.com/science/article/pii/S1 https://etd.auburn.edu/bitstream/handle/10415/1133/Vautier_Mana_34.pdf?sequence=1 https://agupubs.onlinelibrary.wiley.c

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following are the position and velocity in the GCRF frame of satellite
# MOLNIYA 1-83 the 25th of June, 2006 at 00:33:43 UTC.

initialPosition <-c(-14568679.5026116, -4366250.78287623, 9417.9289105405)
initialVelocity <- c(-3321.17428902497, -3205.49400830455, 4009.26862308806)
initialTime <- "2006-06-25 00:33:43"

# Molniya satellites have a mass of approximately 1600 kg and a cross-section of
# 15 m2. Additionally, let´s use 2.2 and 1.2 as approximately values of the
# drag and reflectivity coefficients, respectively.

molniyaMass <- 1600
molniyaCrossSection <- 15
molniyaCr <- 1.2
molniyaCd <- 2.2

# Let´s calculate the position and velocity of the satellite for each minute of
# the following 10 minutes.

targetTimes <- seq(0, 600, by=60)
hpop_results <- hpop(initialPosition, initialVelocity, initialTime, targetTimes,
                     molniyaMass, molniyaCrossSection, molniyaCrossSection,
                     molniyaCr, molniyaCd)
}
```

ITRFtoGCRF *Convert coordinates from ITRF to GCRF*

Description

The ITRF (International Terrestrial Reference Frame) is an ECEF (Earth Centered, Earth Fixed) frame of reference, i.e., a non-inertial frame of reference where the origin is placed at the center of mass of Earth, and the frame rotates with respect to the stars to remain fixed with respect to the Earth surface as it rotates. The Z-axis extends along the true North as defined by the IERS reference pole, and the X-axis extends towards the intersection between the equator and the Greenwich meridian at any time.

The GCRF (Geocentric Celestial Reference Frame) frame of reference is an Earth-centered inertial coordinate frame, where the origin is also placed at the center of mass of Earth and the coordinate

frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The X-axis is aligned with the mean equinox of Earth at 12:00 Terrestrial Time on the 1st of January, 2000, and the Z-axis is aligned with the Earth´s rotation axis.

This function requires the `asteRiskData` package, which can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
ITRFtoGCRF(position_ITRF, velocity_ITRF, dateTime)
```

Arguments

`position_ITRF`	Vector with the X, Y and Z components of the position of an object in ITRF frame, in m.
`velocity_ITRF`	Vector with the X, Y and Z components of the velocity of an object in ITRF frame, in m/s.
`dateTime`	Date-time string with the date and time in UTC corresponding to the provided position and velocity vectors. This specifies the time for which the conversion from ITRF to GCRF coordinates will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of GCRF coordinates refers varies with time due to the motion of Earth.

Value

A list with two elements representing the position and velocity of the satellite in the GCRF (Earth-centered non-intertial) frame of reference. Position values are in m, and velocity values are in m/s. Each of the two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order.

References

https://celestrak.org/columns/v02n01/

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following were the position and velocity of satellite MOLNIYA 1-83
# the 25th of June, 2006 at 00:33:43 UTC in the ECEF frame (in m and m/s).

position_ITRF <- c(1.734019e+06, -1.510972e+07, 39.08228)
velocity_ITRF <- c(1468.832, -3962.464, 4007.039)

# Let´s convert them into the GCRF frame

coordinates_GCRF <- ITRFtoGCRF(position_ITRF, velocity_ITRF, "2006-06-25 00:33:43")
}
```

ITRFtoLATLON *Convert coordinates from ITRF to geodetic latitude, longitude and altitude*

Description

The ITRF (International Terrestrial Reference Frame) is an ECEF (Earth Centered, Earth Fixed) frame of reference, i.e., a non-inertial frame of reference where the origin is placed at the center of mass of Earth, and the frame rotates with respect to the stars to remain fixed with respect to the Earth surface as it rotates. The Z-axis extends along the true North as defined by the IERS reference pole, and the X-axis extends towards the intersection between the equator and the Greenwich meridian at any time. This function converts Cartesian coordinates in the ECEF frame to geodetic latitude, longitude and altitude.

Usage

```
ITRFtoLATLON(position_ITRF, degreesOutput=TRUE)
```

Arguments

position_ITRF	Vector with the X, Y and Z components of the position of an object in ITRF frame, in m.
degreesOutput	Logical indicating if the output should be in sexagesimal degrees. If degreesOutput=FALSE, the output will be in radians.

Value

A vector with three elements, corresponding to the latitude and longitude in degrees (or radians if specified) and the altitude in m.

References

https://arc.aiaa.org/doi/10.2514/6.2006-6753

Examples

```
coordinates_ITRF <- c(5062040.1, -530657.4, 3863936.5)

# Let's calculate the geodetic latitude, longitude and altitude

geodetic <- ITRFtoLATLON <- (coordinates_ITRF)

# A longer application example follows, useful to represent the groundtrack of a
# satellite after propagaton with SGP4/SDP4

if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.
```

```
n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Let´s calculate the position and velocity of the satellite 1 day later

state_1day_TEME <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                         Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)

# We can now convert the results in TEME frame to ITRF frame, previously
# multiplying by 1000 to convert the km output of sgdp4 to m

state_1day_ITRF <- TEMEtoITRF(state_1day_TEME$position, state_1day_TEME$velocity,
                              "2006-06-27 00:58:29.34")

# Finally, we can convert the ECEF coordinates to geodetic latitude, longitude
# and altitude

state_1day_geodetic <- ITRFtoLATLON(state_1day_ITRF$position)
}
```

`JPLephemerides` *Calculate JPL main celestial objects ephemerides for a given Modified Julian Date*

Description

NASA's Jet Propulsion Laboratory (JPL) provides mathematical models of the Solar System known as Development Ephemerides (DE). The models are given as sets of Chebyshev coefficients, which cam be used to calculate the position (and its derivatives) of the Sun, the eight major planets, Pluto and the Moon. This function employes JPL DE440 to calculate the position (and optionally velocities also) of the mentioned celestial objects, in ICRF frame. JPL DE440 covers the period from 1550 to 2650 AC. In addition to the position of celestial objects, lunar libration angles are also calculated. Internally, calculations are performed by employing Clenshaw's algorithm together with the Chebyshev coefficients provided by JPL DE440. The target time should be specified as a Modified Julian Date (MJD). MJD in different time systems can be used. Currently, UTC, UT1, TT and TDB are supported. Additionally, a central body with respect to which positions and velocities are calculated should be specified. By default, the Solar System Barycenter (SSB) is used, but additionally Mercury, Venus, Earth, Moon, Mars, Jupiter, Saturn, Uranus, Neptune or Pluto can be selected. Note that this function requires the additional package asteRiskData, which provides the Chebyshev coefficients, and can be installed by running `install.packages("asteRiskData", repos="https://rafael-ayala.github.io/drat/")`

Usage

```
JPLephemerides(MJD, timeSystem="UTC", centralBody="SSB", derivatives="acceleration")
```

Arguments

MJD	Modified Julian Date of the time for which celestial object ephemerides should be calculated. MJD are fractional number of days since midnight of the 17th of November, 1858. The MJD of a date-time string can be obtained with function dateTimeToMJD.
timeSystem	Time system into which the MJD is provided. Should be one from "UTC" (Coordinated Universal Time; default), "UT1" (Universal Time), "TT" (Terrestrial Time) and "TDB" (Barycentric Dynamical Time).
centralBody	String indicating the celestial object that will be taken as the center of coordinates to which positions and velocities are referred. Must be one of "SSB" (Solar System Barycenter), "Mercury", "Venus", "Earth", "Moon", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune" or "Pluto".
derivatives	String indicating what derivatives of positions should be calculated. Must be one of "none", "velocity" or "acceleration". If "none", only position is calculated. If "velocity", velocities are calculated, as well as first order derivatives of Moon libration angles. If "acceleration", both velocities and accelerations (as well as second order derivatives of Moon libration angles) are calculated.

Value

A list of vectors providing the positions (in meters), velocities (in m/s; only if requested), accelerations (in m/s^2; only if requested), Moon libration angles (in radians), first derivatives of Moon libration angles (in radians/s; only if velocities were requested) and second derivatives of Moon libration angles (in radians/s^2; only if accelerations were requested) of celestial objects with respect to the specified central body. For position, velocity and acceleration vectors, X, Y and Z components are given in this order. For Moon libration angles and their derivatives, they are given in the following order: phi, theta and psi.

References

https://gssc.esa.int/navipedia/index.php/Julian_Date https://gssc.esa.int/navipedia/index.php/Transformations_between_Tim

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# Let's calculate the MJD of the 12th of June, 2000 at 10:00:00 UTC time, in UTC

MJD_UTC <- dateTimeToMJD("2000-06-12 10:00:00", timeSystem = "UTC")

# Let's now calculate the JPL ephemerides using Earth as the central body:

ephemerides <- JPLephemerides(MJD_UTC, timeSystem = "UTC", centralBody="Earth")

# We can now calculate, for example, the exact distance between the barycenters
# of Earth and Moon
```

```
sqrt(sum(ephemerides$positionMoon^2))
}
```

KOEtoBCI *Calculate BCI coordinates from Keplerian orbital elements*

Description

Keplerian orbital elements are a set of six parameters used to described the orbits of celestial objects, including satellites. While satellites do not follow a perfectly Keplerian orbit, their state at any point can be defined by the orbital parameters that they would have if they were located at the same position with the same velocity following a perfectly Keplerian orbit (i.e., if perturbations were absent). These are called osculating orbital elements.

A complete set of six Keplerian elements defines unequivocally the position and velocity of the satellite in a given frame of reference, and therefore can be used to calculate its cartesian coordinates. This function calculates the coordinates of a satellite in an BCI (body-centered inertial) frame of reference from a set of Keplerian orbital elements. The exact BCI frame of the resulting coordinates is the same used to define the supplied orbital elements.

By default, the function assumes the central body is the Earth and uses the GM of the Earth in the TDB frame. This can be changed by providing the GM of the desired central body in m^3/s^2.

Usage

```
KOEtoBCI(a, e, i, M, omega, OMEGA, keplerAccuracy=10e-8, maxKeplerIterations=100,
centralBodyGM=GM_Earth_TDB)
```

Arguments

a	Semi-major axis of orbital ellipse in meters.
e	Numerical eccentricity of the orbit. Eccentricity measures how much the orbit deviates from being circular.
i	Inclination of the orbital plane in radians. Inclination is the angle between the orbital plane and the equator.
M	Mean anomaly of the orbit in radians. Mean anomaly indicates where the satellite is along its orbital path, and is defined as the angle between the direction of the perigee and the hypothetical point where the object would be if it was moving in a circular orbit with the same period as its true orbit after the same amount of time since it last crossed the perigee had ellapsed.
omega	Argument of perigee in radians. This is the angle between the direction of the ascending node and the direction of the perigee (the point of the orbit at which the object is closest to the central body).
OMEGA	Right ascension of the ascending node in radians. This is the angle between the direction of the ascending node (the point where the satellite crosses the equatorial plane moving north) and the direction of the First Point of Aries (which indicates the location of the vernal equinox).

`keplerAccuracy` Accuracy to consider Kepler's equation solved when calculating eccentric anomaly from mean anomaly. If two consecutive solutions differ by a value lower than this accuracy, integration is considered to have converged.

`maxKeplerIterations` Maximum number of iterations after which fixed-point integration of Kepler's equation will stop, even if convergence according to the accuracy criterion has not been reached.

`centralBodyGM` GM of the central body in m^3/s^2. By default, the GM of the Earth in the TDB frame is used.

Value

A list with two elements representing the position and velocity of the satellite in the same BCI (body-centered inertial) frame of reference into which the provided orbital elements were defined. Position values are in m, and velocity values are in m/s. Each of the two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order.

References

https://www.gsc-europa.eu/system-service-status/orbital-and-technical-parameters https://celestrak.org/columns/v02n01/
https://downloads.rene-schwarz.com/download/M001-Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf

Examples

```
# Let's calculate the ECI coordinates from the orbital elements provided by a
# TLE. It should be noted that this is often not recommended, since the orbital
# elements supplied in a TLE are not osculating orbital elements, but instead
# mean orbital elements set to fit a range of actual observations. The
# recommended procedures are to use TLE only in conjunction with the SGP4/SDP4
# models, and viceversa.
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(86400))  # Multiplication by 2pi/86400 to convert to radians/s
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians

# The semi-major axis can be calculated from the mean motion in radians/s
# as follows: (mu is the standard gravitational parameter of Earth)

mu <- 3.986004418e14 # in units of m3 s-2
a0 <- (mu^(1/3))/(n0^(2/3))

# The ECI coordinates can then be calculated. In this case, they will be in TEME
# frame, since the original orbital elements are derived from a TLE
coordinates_ECI <- KOEtoBCI(a0, e0, i0, M0, omega0, OMEGA0)
```

`lambert` *Solve Lambert's problem to determine a transfer orbit*

Description

Given 2 position vectors and a time difference between them, calculates the velocity that the object should have at the initial and final states to follow a transfer orbit in the target time (Lambert's problem). The transfer orbit can be elliptical, parabolic or hyperbolic, all of which are dealt with. In the case of elliptical orbits, multi-revolution orbits are possible. Additionally, low-path and high-path orbits are also possible. All possible solutions are returned for the case of elliptical transfers. The user must specify if a retrograde transfer is desired. By default, prograde transfers are calculated. Currently, the formulation by Dario Izzo is applied to solve Lambert's problem (https://link.springer.com/article/10.1007/s10569-014-9587-y).

Usage

```
lambert(initialPosition, finalPosition, initialTime, finalTime, retrogradeTransfer=FALSE,
        centralBody="Earth", maxIterations=2000, atol=0.00001, rtol=0.000001)
```

Arguments

`initialPosition`
: Vector with the 3 components of the initial position in Cartesian coordinates, in meters.

`finalPosition`
: Vector with the 3 components of the initial position in Cartesian coordinates, in meters.

`initialTime`
: Either date-time string in UTC indicating the time corresponding to the initial state vector of the satellite, or numeric value indicating the starting time in seconds since an arbitrary reference instant. If provided as a date-time string in UTC, `finalTime` can be provided either as another date-time string in UTC (in which case the transfer time will be determined as the difference between the 2 date-time strings), or as a numeric value indicating the seconds ellapsed since `initialTime`. If provided as a numeric value, `finalTime` can only be provided as another numeric value indicating the number of seconds since the same arbitrary reference for objects in deep space, and also for objects near Earth if `targetTime` is provided as a date-time string.

`finalTime`
: Either date-time string in UTC indicating the time corresponding to the final state vector of the satellite, or numeric value indicating the final time in seconds. If `initialTime` was provided as a date-time string in UTC, then `finalTime` can be provided as either of the two (date-time string or numeric value in seconds). In this case, providing `finalTime` as a numeric value will be interpreted as the number of seconds since the instant specified for `initialTime`. If `initialTime` was provided as a numeric value (indicating the time in seconds since an arbitrary reference point), then `finalTime` can only be provided as another numeric value, which will be interpreted as the number of seconds since the same reference instant as that used for `initialTime`, and therefore the transfer time will be calculated as the difference between `initialTime` and `finalTime`.

`retrogradeTransfer`
: Logical indicating if retrograde transfer orbits should be calculated, i.e., transfer orbits with an inclination higher than 180º. By default, `retrogradeTransfer=FALSE`, and prograde transfer orbits are calculated.

`centralBody`
: String indicating the central body around which the satellite is orbiting. Can be one of `c("Sun", "Mercury", "Venus", "Earth", "Moon", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")` (case insensitive).

`maxIterations`
: Maximum number of iterations to perform at the different root-finding steps when solving Lambert's problem.

`atol`
: Absolute tolerance value used for the convergence criterion at the different root-finding steps when solving Lambert's problem.

`rtol`
: Relative tolerance value used for the convergence criterion at the different root-finding steps when solving Lambert's problem.

Value

A list with a number of elements equal to the number of possible transfer orbits found. For hyperbolic and parabolic transfer orbits, there will always be a single possible transfer orbit. For elliptic transfer orbits, multi-revolution orbits may exist if the transfer time is large enough. If they are possible, they will be calculated together with the basic, single-revolution orbit. Each element of the top-level list is in itself a list with the following elements:

`numberRevs`
: Number of complete revolutions that will be performed in this transfer orbit. Always equals 0 for parabolic and hyperbolic transfer orbits, as well as for the non-multirevolution orbit of elliptic transfers.

`path`
: String indicating if the transfer orbit is of the high-path type (i.e., has its second focus located beyond the vector connecting the initial and final positions) or of the low-path type (second focus located between this vector and the first focus).

`orbitType`
: String indicating the type of transfer orbit (elliptic, parabolic or hyperbolic).

`v1`
: Velocity vector in m/s that the object should have at the start of the transfer orbit.

`v2`
: Velocity vector in m/s that the object should have at the end of the transfer orbit.

References

https://link.springer.com/article/10.1007/s10569-014-9587-y

Examples

```
# Consider the following initial and final positions:
initialPosition <- c(15945.34, 0, 0) * 1000
finalPosition <- c(12214.83899, 10249.46731, 0) * 1000
# Given a time difference of 76 minutes between the 2 states, calculate the
# velocity that the spacecraft should have at the beginning and end of the
# transfer orbit
lambertSolution <- lambert(initialPosition, finalPosition, 0, 76*60)
length(lambertSolution)
lambertSolution[[1]]$orbitType
lambertSolution[[1]]$v1
```

```
lambertSolution[[1]]$v2

# A single transfer orbit is possible (elliptic, single-revolution orbit)
```

LATLONtoGCRF *Convert coordinates from geodetic latitude, longitude and altitude to GCRF*

Description

The GCRF (Geocentric Celestial Reference Frame) frame of reference is an Earth-centered inertial coordinate frame, where the origin is also placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The X-axis is aligned with the mean equinox of Earth at 12:00 Terrestrial Time on the 1st of January, 2000, and the Z-axis is aligned with the Earth´s rotation axis. This function converts geodetic latitude, longitude and altitude to Cartesian coordinates in the GCRF frame. The WGS84 Earth ellipsoid model is used.

Usage

```
LATLONtoGCRF(position_LATLON, dateTime, degreesInput=TRUE)
```

Arguments

position_LATLON	Vector with the latitude, longitude and altitude of the object. Latitude and longitude can be provided in sexagesimal degrees or in radians (by default, sexagesimal degrees are asumed). Altitude must be provided in meters.
dateTime	Date-time string with the date and time in UTC corresponding to the provided geodetic coordinates.
degreesInput	Logical indicating if the input latitude and longitude are in sexagesimal degrees. If degreesInput=FALSE, the input will be considered to be in radians. This specifies the time for which the conversion from geodetic coordinates to GCRF will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of GCRF coordinates corresponds varies with time due to the motion of Earth.

Value

A vector with three elements, corresponding to the X, Y and Z components of position in meters in the ECEF frame, in this order.

References

https://apps.dtic.mil/sti/pdfs/ADA280358.pdf

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
latitude <- 37.3891
longitude <- -5.9845
altitude <- 20000

# Let´s calculate the corresponding coordinates in GCRF frame, assuming that
# the provided coordinates were valid the 20th of October, 2020 at 15:00:00 UTC

coordinatesGCRF <- LATLONtoGCRF(c(latitude, longitude, altitude),
                                dateTime="2020-10-20 15:00:00")
}
```

LATLONtoITRF *Convert coordinates from geodetic latitude, longitude and altitude to ITRF*

Description

The ITRF (International Terrestrial Reference Frame) is an ECEF (Earth Centered, Earth Fixed) frame of reference, i.e., a non-inertial frame of reference where the origin is placed at the center of mass of Earth, and the frame rotates with respect to the stars to remain fixed with respect to the Earth surface as it rotates. The Z-axis extends along the true North as defined by the IERS reference pole, and the X-axis extends towards the intersection between the equator and the Greenwich meridian at any time. This function converts geodetic latitude, longitude and altitude to Cartesian coordinates in the ITRF frame. The WGS84 Earth ellipsoid model is used.

Usage

```
LATLONtoITRF(position_LATLON, degreesInput=TRUE)
```

Arguments

position_LATLON	Vector with the latitude, longitude and altitude of the object. Latitude and longitude can be provided in sexagesimal degrees or in radians (by default, sexagesimal degrees are asumed). Altitude must be provided in meters.
degreesInput	Logical indicating if the input latitude and longitude are in sexagesimal degrees. If degreesInput=FALSE, the input will be considered to be in radians.

Value

A vector with three elements, corresponding to the X, Y and Z components of position in meters in the ITRF frame, in this order.

References

https://apps.dtic.mil/sti/pdfs/ADA280358.pdf

Examples

```
latitude <- 37.3891
longitude <- -5.9845
altitude <- 20000

# Let´s calculate the corresponding coordinates in ECEF frame

coordinates_ITRF <- LATLONtoITRF(c(latitude, longitude, altitude))
```

`listAvailablePlanetaryEphemerides` *List Available Planetary Ephemerides Files*

Description

This function lists all available JPL DE ephemerides files (with .bsp extension). More JPL DE ephemerides files can be downloaded from the JPL Horizons website (https://ssd.jpl.nasa.gov/?horizons), and registered with the `registerPlanetaryEphemerides`.

Usage

```
listAvailablePlanetaryEphemerides()
```

Arguments

None

Value

Returns a vector of filenames of the available JPL DE ephemerides files.

References

NAIF/SPICE Documentation: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html

Examples

```
# List available ephemerides files
listAvailablePlanetaryEphemerides()
```

parseTLElines *Parse the lines of a TLE*

Description

TLE (Two-/Three- Line Element) is the standard format for representing orbital state vectors. This function parses a character vector where each element represents a line of the TLE. The supplied character vector can have either 2 (for Two Line Elements) or 3 (for Three Line Elements) elements. The two lines of a Two Line Element contain all the information. The additional line in a Three Line Element is optional, and contains just the satellite name. For a detailed description of the TLE format, see https://celestrak.org/columns/v04n03/#FAQ01.

Usage

```
parseTLElines(lines)
```

Arguments

lines	Character vector where each element is a string corresponding to a line of the TLE. The character vector must have either 2 or 3 elements.

Value

A list with the following elements that define the orbital state vector of the satellite:

NORADcatalogNumber	NORAD Catalog Number, also known as Satellite Catalog Number, assigned by United States Space Command to each artificial object orbiting Earth
classificationLevel	Classification level of the information for the orbiting object. Can be unclassified, classified, secret or unknown
internationalDesignator	International Designator, also known as COSPAR ID, of the object. It consists of the launch year, separated by a hyphen from a three-digit number indicating the launch number for that year and a set of one to three letters indicating the piece for a launch with multiple pieces.
launchYear	The launch year of the object
launchNumber	The launch number of the object during its launch year
launchPiece	The piece for the launch of the object, if it was a launch with multiple pieces
dateTime	Date time string to which the orbital state vector corresponds
elementNumber	Element number for the object. In principle, every time a new TLE is generated for an object, the element number is incremented, and therefore element numbers could be used to assess if all the TLEs for a certain object are available. However, in practice it is observed that this is not always the case, with some numbers skipped and some numbers repeated.

`inclination` Mean orbital inclination of the satellite in degrees. This is the angle between the orbital plane of the satellite and the equatorial plane

`ascension` Mean longitude of the ascending node of the satellite at epoch, also known as right ascension of the ascending node, in degrees. This is the angle between the direction of the ascending node (the point where the satellite crosses the equatorial plane moving north) and the direction of the First Point of Aries (which indicates the location of the vernal equinox)

`eccentricity` Mean eccentricity of the orbit of the object. Eccentricity is a measurement of how much the orbit deviates from a circular shape, with 0 indicating a perfectly circular orbit and 1 indicating an extreme case of parabolic trajectory

`perigeeArgument`
Mean argument of the perigee of the object in degrees. This is the angle between the direction of the ascending node and the direction of the perigee (the point of the orbit at which the object is closest to the Earth)

`meanAnomaly` Mean anomaly of the orbit of the object in degrees. This indicates where the satellite is along its orbital path. It is provided as the angle between the direction of the perigee and the hypothetical point where the object would be if it was moving in a circular orbit with the same period as its true orbit after the same amount of time since it last crossed the perigee had ellapsed. Therefore, 0 denotes that the object is at the perigee

`meanMotion` Mean motion of the satellite at epoch in revolutions/day

`meanMotionDerivative`
First time derivative of the mean motion of the satellite in revolutions/day^2^

`meanMotionSecondDerivative`
Second time derivative of the mean motion of the satellite in revolutions/day^3^.

`Bstar` Drag coefficient of the satellite in units of (earth radii)^-1^. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.

`ephemerisType` Source for the ephemeris (orbital state vector). Most commonly, it is distributed data obtained by combaining multiple observations with the SGP4/SDP4 models

`epochRevolutionNumber`
Number of full orbital revolutions completed by the object

`objectName` Name of the object, retrieved from the first line of the TLE if a Three Line Element was provided

References

https://celestrak.org/columns/v04n03/#FAQ01

Examples

```
# The following lines correspond to a TLE for Italsat 2 the 26th of June, 2006
# at 00:58:29.34 UTC.

italsat2_lines <- c("ITALSAT 2",
"1 24208U 96044A   06177.04061740 -.00000094  00000-0  10000-3 0  1600",
```

```
"2 24208   3.8536  80.0121 0026640 311.0977  48.3000  1.00778054 36119")

italsat2_TLE <- parseTLElines(italsat2_lines)
italsat2_TLE
```

`rad2deg` *Converts an angle in radians to degrees*

Description

This function converts an angle in radians to degrees.

Usage

```
rad2deg(radians)
```

Arguments

`radians`	Value of the angle in radians.

Value

The corresponding value of the angle in degrees.

Examples

```
rad2deg(pi)
```

`readBinDAF` *Read a generic binary DAF file*

Description

DAF (Double Precision Array File) is a binary file architecture designed to store arrays of double precision arrays used by SPICE, NAIF toolkit software library. The architecture forms the basis of multiple file formats used to store different data related to astrodynamics, such as SPK, PCK and CK files (https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html).

DAF files provide a generic architecture onto which more specific file formats are implemented. They are organized in records of fixed length (1024 bytes) containing different information. The first record is called the file record, and contains global metadata for the file. This is followed by an optional block comprising any number of comment records. After this, the file consits of sets of summary records, name records and element records. These are structured as blocks of 1 summary record (which contains multiple array summaries, providing metadata about each array), followed by 1 name record (comprising names for the corresponding arrays whose summaries were in the previous summary record) and finally by as many element records as required to store the arrays

described in the corresponding summary records. For a detailed description of the DAF architecture, see NAIF documentation (https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html).

This function allows to read any binary file with a DAF architecture in a generic way, without applying any specific formatting to its contents. Note that this will result in just a list of the file global metadata, comments and different arrays (each one with a summary, a name and a set of elements). The number of elements and meaning of each included in each array, as well as the descriptor integers and doubles contained in the summaries, vary for each specific file type. Therefore, reading a DAF file generically is not likely to bring much meaningful information, unless a precise understanding of the specific file format is taken into account later.

Usage

```
readBinDAF(filename)
```

Arguments

`filename`	Path to the binary DAF file.

Value

A list with three elements. The first element, named `metadata`, corresponds to the "file record", which is always the first record in a DAF file, and is a list with the following metadata elements:

`fileType`	String of the format DAF/XXXX indicating the specific file format for the DAF file
`numDoublesSummary`	Number of double precision numbers in each array summary
`numIntsSummary`	Number of integers in each array summary
`summaryRecordSizeDoubles`	Size of each array summary in doubles
`numCharsName`	Number of characters in each array name
`description`	String with an internal name or description of the DAF file
`firstSummaryRecNum`	Integer indicating which is the 1st summary record This can be used to infer the number of comment files. For example, a value of 3 indicates that the 3rd record is the 1st summary record. Since the 1st record is always the file record, this means there is 1 comment record
`lastSummaryRecNum`	Integer indicating which is the last summary record
`firstFreeAddress`	Integer indicating the first free address (in bytes) of the file, i.e., the address of the last byte of the last element record plus 1
`endianString`	String indicating the endianness of the file. This is automatically taken into account when reading the file. It can be either LTL-IEEE (little endianness) or BIG-IEEE (big endianness), and is determined by the architecture of the system where the file was written

`ftpString` String used to verify integrity of the DAF file. It should be exactly equal to `"FTPSTR:\r:\n:\r\n:\r:\x81:\020\xce:ENDFTP"`

The second element is named `comments`, and is a character vector where each element is a line of comments.

The third element is named `arrays`, and is a nested list where each top-level element represents one of the arrays stored in the DAF file and its associated metadata. Each of the top-level elements is itself a list with the following 3 elements:

`arrayName` String with the name of the array

`arraySummary` A list with the multiple doubles and integers that are stored in each array summary of summary records and which provide metadata describing each array. The elements are named as Double1, Double2, ..., DoubleN; Integer1, Integer2, ..., Integer(M-2) and finally initialArrayAddress and finalArrayAddress. N and M are respectively the number of doubles and integers in each array summary, and are given in elements `numDoublesSummary` and `numIntsSummary` of the `metadata` element of the top-level list. Note that the number of doubles and integers describing each array, as well as the meaning of each, varies between different specific file formats, and therefore no exact meaning can be derived when simply reading the file as a generic DAF file. The exception to this are the last 2 integers, which always are respectively the initial and final addresses of the elements corresponding to the array within the DAF file, in double precision numbers (and therefore, in order to obtain byte addresses it must be multiplied by 8 and subtract 7)

`arrayElements` A numeric vector with all the elements of the array. Note that this includes potentially constants, actual data and additional array metadata. Furthermore, the number, order and meaning of the elements differs greatly between different specific types and subtypes of DAF files, and therefore it is hard to extract any meaningful information without knowledge of the internal organization of each array

References

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/daf.html

Examples

```
# The file vgr2_jup230.bsp provided with the package includes information for the
# Jupiter flyby of Voyager 2

testDAF <- readBinDAF(paste0(path.package("asteRisk"), "/vgr2_jup230.bsp"))
testDAF$metadata
# The file seems to be of type SPK
testDAF$comments
length(testDAF$arrays)
# It contains a single array
```

`readBinSPK` *Read a binary SPK file*

Description

SPK (Spacecraft and Planet Kernel) is a binary file format developed by NAIF to store ephemerides (trajectory) of celestial bodies and spacecraft in the Solar System. The file format is based on the DAF architecture (see readBinDAF). A detailed description of the SPK file format can be found in NAIF's documentation (https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html).

Each SPK file contains several segments, with each segment comprising a summary (or descriptor), a name and an array of double precision elements. Each segment is conceptually equivalent to an array in the context of generic DAF files. There are several types of SPK segments defined by NAIF, each identified by an SPK type code currently ranging from 1 to 21 (some intermediate values are not used or not available for general public use). Each segment type provides ephemerides information in a different way. Note that the segments stored in a single SPK file can be of different types. A detailed description of the organization of the arrays for each SPK type can be found at https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html

This function allows to read SPK binary files of all types except 4, 6, 7 and 16. The data will be presented properly formatted and the meaning of each element is assigned during the reading process of the file. It should be noted that this function just reads SPK kernels; it does not provide any evaluation of ephemerides at arbitrary target times. The process of performing such evaluation differs between SPK types. For example, types 2, 3, 14 and 20 require Chebyshev interpolation; typees 8 and 9 require Lagrange interpolation; type 10 requires the aplication of SGP4/SDP4, etc. Nevertheless, it is still possible to obtain direct ephemerides information just by reading the SPK kernels since many of the types often include reference state vectors between which interpolation is applied, but that can be directly used at the epochs corresponding to said reference state vectors.

Usage

```
readBinSPK(filename)
```

Arguments

`filename`	Path to the binary SPK file.

Value

A list with two elements. The first element, named `comments`, is a character vector where each element is a line of comments.

The second element is named `segments`, and is a nested list where each top-level element represents one of the segments stored in the SPK file and its associated metadata. Each of the top-level elements is itself a list with the following 3 elements:

segmentName : String with the name of the segment

segmentSummary : A list with the multiple doubles and integers that are stored in each array summary of summary records and which provide metadata describing each array. In the case of SPK files, these are always the following 9 elements:

SPKType : A description of the type of SPK segment

initialEpoch : The initial epoch for the interval for which ephemeris data are provided in the segment, in ephemeris seconds (equivalent to TDB seconds) past Julian year 2000

finalEpoch : The initial epoch for the interval for which ephemeris data are provided in the segment, in ephemeris seconds (equivalent to TDB seconds) past Julian year 2000

targetNAIFCode : The NAIF integer code for the object for which the segment provides ephemerides. For details, see https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif_ids.html

targetNAIFCode : The NAIF integer code for the central body of the reference frame in which the segment provides ephemerides. For details, see https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif

frameNAIFCode : The NAIF frame code for the reference frame in which the segment provides ephemerides. For details, see https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/frames.html

SPKType : The SPK type code for the segment

initialArrayAddress : The initial address of the array elements corresponding to this segment within the SPK file, in double precision numbers (in order to obtain byte address, multiply by 8 and subtract 7)

finalArrayAddress : The final address of the array elements corresponding to this segment within the SPK file, in double precision numbers (in order to obtain byte address, multiply by 8 and subtract 7)

segmentData : A list with the actual ephemeris data contained in the segment, as well as some type-specific additional metadata

The contents of the last element, `segmentData`, are different for each SPK type. Here a summary is provided for each one, but for more detailed descriptions see NAIF's documentation.

For type 1, which provide Modified Difference Arrays (MDAs), a list where each element is one of the records of the segment. Each of these elements contains the following elements:

`referenceEpoch` The reference epoch that should be used when using this MDA to compute a state vector

`referenceEpoch` The final epoch for which this MDA should be used to compute a state vector

`stepsizeFunctionVector`
A vector of differences between the reference point and the epochs of all the other data points used to fit the interpolation polynomials from which the MDA is derived. This is of length 15

`referencePosition`
A numeric vector of length 3 containing the X, Y and Z components of the position at the reference epoch, in km

`referencePosition`
A numeric vector of length 3 containing the X, Y and Z components of the velocity at the reference epoch, in km/s

`MDA` A matrix with 15 rows and 3 columns providing the constant coefficients to interpolate position, velocity and acceleration at a target epoch. The 1st, 2nd and 3rd columns give the coefficients for the X, Y and Z components respectively. The given coefficients basically are the coefficients of the interpolation polynomial when expressed in Modified Divided Differences (MDD) form. For details, see Shampine and Gordon, 1975. Note that even though the matrix will always have 15 rows (corresponding to 15 coefficients for each components, or an interpolation polynomial degree of 14), some of these can have a value of 0 up to

the 15th row, effectively leading to an interpolation polynomial of degree lower than 14.

`maxIntegrationOrderP1`
: The maximum order for the interpolation polynomial that should be applied amongst all 3 components plus 1. For example, if the interpolation polynomial orders for the X, Y and Z components are 6, 8 and 7 respectively, this element will have a value of 9 (`(max(c(6,8,7))+1)`).

`integrationOrderArray`
: A vector of 3 integers indicating the order of the interpolation polynomials that should be applied for the X, Y and Z components of acceleration respectively.

A brief description is provided here for the meaning of the coefficients, based on the 'SPICE spke01 math' monograph by Robert Werner. MDAs are a modified version of the more standard coefficients obtained through the divided differences method, which represent an interpolation polynomial in Newton form. In order to calculate a position and velocity, first the basis functions must be computed for both position and velocity. Then each of the components of the basis functions (each of an increasing degree) must be multiplied by the corresponding MDA coefficient, and all the resulting terms are summed.

For type 2, which provide Chebyshev coefficients for position only and at equally spaced time steps, a list with the following elements:

polynomialDegree : An integer indicating the order of the interpolation polynomial that should be applied for all the components.

chebyshevCoefficients : A matrix where each row corresponds to an interpolation interval, and with the following columns:

- **initialEpoch** : Initial epoch of the interpolation intervals, in ephemeris (TDB) seconds since J2000
- **midPoint** : Epoch for the midpoint of the interpolation intervals, in ephemeris (TDB) seconds since J2000
- **intervalRadius** : Radius of the interpolation intervals, in seconds)
- **positionXCoeffi** : A set of N columns, with i ranging from 1 to N, providing the Chebyshev coefficients for the X component of the position. N is the number of coefficients for each component, which is equal to `polynomialDegree + 1`
- **positionYCoeffi** : As `positionXCoeffi`, but for the Y component of position.
- **positionZCoeffi** : As `positionXCoeffi`, but for the Z component of position.

For type 3, which provide Chebyshev coefficients for position and velocity and at equally spaced time steps, the same as for type 2, but `chebyshevCoefficients` contains the following additional elements:

`velocityXCoeffi`
: A set of N columns, with i ranging from 1 to N, providing the Chebyshev coefficients for the X component of the velocity. N is the number of coefficients for each component, which is equal to `polynomialDegree + 1`

`velocityYCoeffi`
: As `velocityXCoeffi`, but for the Y component of velocity.

`velocityZCoeffi`
: As `velocityXCoeffi`, but for the Z component of velocity.

For type 5, which provide discrete state vectors to be propagated following the laws of two-body motion, a list with the following elements:

centralBodyGM The GM parameter (gravitational constant) for the central body, in cube kilometers per square seconds

stateVectors A matrix where each row corresponds to a state vector, and with the following columns:

epoch : Epoch of the state vectors intervals, in ephemeris (TDB) seconds since J2000

positionX : X component of the position, in km

positionY : Y component of the position, in km

positionZ : Z component of the position, in km

velocityX : X component of the velocity, in km/s

velocityY : Y component of the velocity, in km/s

velocityZ : Z component of the velocity, in km/s

For type 8, which provide discrete state vectors at equally spaced time steps to which Lagrange interpolation should be applied to obtain state vectors at arbitrary target times, a list with the following elements:

polynomialDegree : The degree of the interpolation polynomial that should be applied

stateVectors : A matrix where each row corresponds to a state vector, and with the following columns:

epoch : Epoch of the state vectors intervals, in ephemeris (TDB) seconds since J2000

positionX : X component of the position, in km

positionY : Y component of the position, in km

positionZ : Z component of the position, in km

velocityX : X component of the velocity, in km/s

velocityY : Y component of the velocity, in km/s

velocityZ : Z component of the velocity, in km/s

For type 9, which provide discrete state vectors at unequally spaced time steps to which Lagrange interpolation should be applied to obtain state vectors at arbitrary target times, the same as for type 8.

For type 10, which provide TLE that should be propagated with SGP4/SDP4, a list with the following elements:

constants : A list with the following constants used by SGP4/SDP4:

J2 : J2 parameter (dimensionless)

J3 : J3 parameter (dimensionless)

J4 : J4 parameter (dimensionless)

sqrtGM : Square root of the GM parameter, where GM is in cube Earth radii per square minutes

highAltBound : High altitude boundary for atmospheric model (in km)

lowAltBound : Low altitude boundary for atmospheric model (in km)

earthRadius : Equatorial radius of Earth (in km)

distUnitsPerRadius : Distance units per Earth radius. This is usually 1. If different than 1, interprete results with caution

TLEs : A matrix where each row corresponds to a TLE, and with the following columns (nutation angles and their rates are not present in some old SPK files, in which case they will have NULL values):

epoch : Epoch of the TLE, in ephemeris (TDB) seconds since J2000

meanMotionDerivative : First derivative of the mean motion in radians/min^2

meanMotionSecondDerivative : Second derivative of the mean motion in radians/min^3

Bstar : Drag coefficient of the satellite in units of (earth radii)^-1. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.

inclination : Mean orbital inclination of the satellite in radians

ascension : Mean longitude of the ascending node of the satellite at epoch, also known as right ascension of the ascending node, in radians

eccentricity : Mean eccentricity of the orbit of the object

perigeeArgument : Mean argument of the perigee of the object in radians

meanAnomaly : Mean anomaly of the orbit of the object in radians

meanMotion : Mean motion of the satellite at epoch in radians/min

deltaPsi : Obliquity (psi angle) of the nutation at epoch, in radians

deltaEpsilon : Longitude (epsilon angle) of the nutation at epoch, in radians

deltaPsiDerivative : Derivative of the obliquity (psi angle) of the nutation at epoch, in radians/second

deltaEpsilonDerivative : Derivative of the longitude (epsilon angle) of the nutation at epoch, in radians/second

For type 12, which provide discrete state vectors at equally spaced time steps to which Hermite interpolation should be applied to obtain state vectors at arbitrary target times, the same as for type 8, but the list contains an additional element:

`windowSize` The window size that should be applied during interpolation

For type 13, which provide discrete state vectors at unequally spaced time steps to which Hermite interpolation should be applied to obtain state vectors at arbitrary target times, the same as for type 12.

For type 14, which provide Chebyshev coefficients for position and velocity and at unequally spaced time steps, the same as for type 3.

For type 15, which provide elements for calculation of ephemerides through the application of a precessing conic propagation model, a list with the following elements:

`epochPeriapsis` Epoch of the periapsis passage in ephemeris (TDB) seconds since J2000

`unitVectorTrajectoryPoleX`
X component of the unit trajectory pole vector, in km

`unitVectorTrajectoryPoleY`
Y component of the unit trajectory pole vector, in km

`unitVectorTrajectoryPoleZ`
Z component of the unit trajectory pole vector, in km

`unitVectorPeriapsisX`
: X component of the unit periapsis vector, in km

`unitVectorPeriapsisY`
: Y component of the unit periapsis vector, in km

`unitVectorPeriapsisZ`
: Z component of the unit periapsis vector, in km

`semiLatusRectum`
: Semi-latus rectum, in km

`eccentricity`
: Eccentricity of the orbit

`J2ProcessingFlag`
: Flag indicating what J2 corrections should be applied when propagating. If 1, regress line of nodes only. If 2, precess line of apsides only. If 3, don't use any corrections. For any other values, regress line of nodes and precess line of apsides

`unitVectorCentralBodyPoleX`
: X component of the unit central body pole vector, in km

`unitVectorCentralBodyPoleY`
: Y component of the unit central body pole vector, in km

`unitVectorCentralBodyPoleZ`
: Z component of the unit central body pole vector, in km

`centralBodyGM`
: The GM parameter (gravitational constant) for the central body, in cube kilometers per square seconds

`centralBodyJ2`
: The J2 parameter for the central body (dimensionless)

`centralBodyRadius`
: Radius of the central body, in km

For type 17, which provide equinoctial elements modelling an object following an elliptic orbit with precessing line of nodes and argument of periapse relative to the equatorial frame of a central body, a list with the following elements:

`epochPeriapsis`
: Epoch of the periapsis passage in ephemeris (TDB) seconds since J2000

`semiMajorAxis`
: Semi-major axis of the orbit, in km

`equinoctialH`
: Value of the equinoctial parameter H at epoch

`equinoctialK`
: Value of the equinoctial parameter K at epoch

`meanLongitude`
: Mean longitude of the orbit at epoch, in radians

`equinoctialP`
: Value of the equinoctial parameter P

`equinoctialQ`
: Value of the equinoctial parameter Q

`longitudePeriapsisDerivative`
: Derivative of the longitude of periapse at epoch (but it is assumed to be constant at other times), in radians/s

`meanLongitudeDerivative`
: Derivative of the mean longitude at epoch (but it is assumed to be constant at other times), in radians/s

`longitudeAscendingNodeDerivative`
: Derivative of the longitude of the ascending node at epoch, in radians/s

`equatorialPoleRightAscension`
: Right ascension of the pole of the orbital reference system relative to the reference frame of the corresponding SPK segment, in radians

`equatorialPoleDeclination`
: Declination of the pole of the orbital reference system relative to the reference frame of the corresponding SPK segment, in radians

For type 18, which provide ephemerides in the format used by ESA on the Mars Express, Rosetta, SMART-1 and Venus Express missions (although applicable to any other object), there are 2 different subtypes: subtype 0 and subtype 1. Subtype 0 should be used to perform sliding-window Hermite interpolation of position and velocity independently. Subtype 1 should be used to perform sliding-window Lagrange interpolation of position and velocity independently. In both cases, `segmentData` is a list with the following elements:

subTypeCode : Subtype code for the type 18 SPK segment

polynomialDegree : An integer indicating the order of the interpolation polynomial that should be applied for all the components.

interpolationType : Type of the interpolation that should be applied. Hermite for subtype 0, and Lagrange for subtype 1

windowSize : The window size that should be applied during interpolation

meanLongitude : Mean longitude of the orbit at epoch, in radians

stateVectors : A matrix where each row corresponds to a state vector. The columns differ depending on the subtype. The following will always be present:

epoch : Epoch of the state vectors intervals, in ephemeris (TDB) seconds since J2000

positionX : X component of the position, in km

positionY : Y component of the position, in km

positionZ : Z component of the position, in km

The following are present for subtype 0:

firstVelocityX : X component of the first velocity value, in km/s. The first velocity value should be used together with the reference position to interpolate position values

firstVelocityY : Y component of the first velocity value, in km/s

firstVelocityZ : Z component of the first velocity value, in km/s

secondVelocityX : X component of the second velocity value, in km/s. The second velocity value should be used together with the reference acceleration to interpolate velocity values

secondVelocityY : Y component of the second velocity value, in km/s

secondVelocityZ : Z component of the second velocity value, in km/s

accelerationX : X component of the acceleration, in km/s^2

accelerationY : Y component of the acceleration, in km/s^2

accelerationZ : Z component of the acceleration, in km/s^2

The following are present for subtype 1:

velocityX : X component of the velocity, in km/s

velocityY : Y component of the velocity, in km/s

velocityZ : Z component of the velocity, in km/s

For type 19, which provides the same data as type 18 but condensing multiple type 18 segments into a single 19 segments (only possible if all the segments have the same target object, central body and reference frame; additionally, the coverage of the segments must overlap only at endpoints and leave no gaps), a list with the following 2 elements:

boundaryChoiceFlag : A flag indicating which minisegment should be used for the epochs at which 2 minisegments overlap. If 0, the earlier minisegment that ends at that epoch is used. If 1, the later minisegment that begins at that epoch is used

minisegments : A nested list where each top-level element represents a type 19 subsegment (called minisegments in NAIF's documentation), each of which is a list with the same elements as a `segmentData` for a type 18 SPK segment, plus the following 2 additional elements:

intervalStartEpoch : Beginning of the interpolation interval covered by this minisegment, in ephemeris (TDB) seconds since J2000

intervalEndEpoch : End of the interpolation interval covered by this minisegment, in ephemeris (TDB) seconds since J2000 Furthermore, a third subtype can be found in type 19 minisegments which is not found for type 18 segments (at least according to NAIF's documentation). This has subtype code 2, and should be used to perform sliding-window Hermite interpolation of position and velocity together (note the difference with subtype 0, where position and velocity are interpolated independently). In this case, element `interpolationType` has a value of "Hermite-joint", and the element describing the minisegment contains the same elements as a type 18 SPK segment of subtype 1 (and not of subtype 0).

For type 20, which provide which provide Chebyshev coefficients for velocity only and at equally spaced time steps together with a reference position (so that position can be interpolated by integration of velocity), a list with the following elements:

polynomialDegree : An integer indicating the order of the interpolation polynomial that should be applied for all the components.

dScale : Distance scale used for both position and velocity, in km. For example, if dScale has a value of 149597870.7 (the length of an astronomical unit, AU, in km), it means the distance units are AU.

tScale : Time scale used for velocity, in TDB seconds. For example, a value of 1 means we are using velocity directly in TDB seconds. A value of 86400 means the time units of velocity would be TDB Julian days, etc.

chebyshevCoefficients : A matrix where each row corresponds to an interpolation interval, and with the following columns:

initialEpoch : Initial epoch of the interpolation intervals, in ephemeris (TDB) seconds since J2000

midPoint : Epoch for the midpoint of the interpolation intervals, in ephemeris (TDB) seconds since J2000

intervalRadius : Radius of the interpolation intervals, in seconds)

velocityXCoeffi : A set of N columns, with i ranging from 1 to N, providing the Chebyshev coefficients for the X component of the velocity. N is the number of coefficients for each component, which is equal to `polynomialDegree + 1`

velocityYCoeffi As `velocityXCoeffi`, but for the Y component of velocity.

velocityZCoeffi As `velocityXCoeffi`, but for the Z component of velocity.

midPointPositionX : X component of the reference position, valid at the midpoint of the interpolation interval. Should be used to interpolate position through integration.

midPointPositionY As `midPointPositionX`, but for the Y component of velocity.

midPointPositionZ As `midPointPositionX`, but for the Z component of velocity.

For type 21, which, like type 1, also provide MDAs, the same nested list as for type 1, with 2 differences. Firstly, unlike for type 1, MDAs of type 21 segments are not limited to a maximum of 15 coefficients per component. Second, each top-level element of the nested list contains an additional element:

`numberCoefficients`
: The number of coefficients provided for the component with the highest interpolation order

References

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif_ids.h
https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/frames.html Shampine, L. F. and Gordon, M. K., Computer Solution of Ordinary Differential Equations: The Initial Value Problem, 1975 Robert Werner, SPICE spke01 math, 2022. https://doi.org/10.5270/esa-tyidsbu

Examples

```
# The file vgr2_jup230.bsp provided with the package includes information for the
# Jupiter flyby of Voyager 2

testSPK <- readBinSPK(paste0(path.package("asteRisk"), "/vgr2_jup230.bsp"))
length(testSPK$segments)
# It contains a single segment
testSPK$segments[[1]]$segmentSummary$SPKType
testSPK$segments[[1]]$segmentSummary$SPKTypeCode
# The segment is of type 1, containing Modified Difference Arrays
length(testSPK$segments[[1]]$segmentData)
# It contains 566 MDAs
```

`readGLONASSNavigationRINEX` *Read a RINEX navigation file for GLONASS satellites*

Description

RINEX (Receiver Independent Exchange Format) is one of the most widely used formats for providing data of satellite navigation systems. The RINEX standard defines several structured text file types, among which navigation files are used to distribute positional information of the satellites. The exact information provided in a RINEX navigation file varies for each satellite navigation system. This function reads RINEX navigation files for satellites of the GLONASS constellation, operated by Russia.

Usage

```
readGLONASSNavigationRINEX(filename)
```

Arguments

filename	Path to the GLONASS RINEX navigation file.

Value

A list with two elements. The first element, named header, is a list with the information contained in the header of the RINEX file. For files using RINEX major version 2, it contains the following elements:

rinexVersion	Version of the RINEX format used in the file
rinexFileType	Type of RINEX file
generatorProgram	Program used to generate the RINEX file
generatorEntity	Individual or organization that generated the file
fileCreationDateString	Date-time string indicating when the file was created
refYear	Reference year for system time correction
refMonth	Reference month for system time correction
refDay	Reference day for system time correction
sysTimeCorrection	Correction to system time scale to fine-tune GLONASS time to UTC in seconds. Since GLONASS time is linked to UTC, it should be a very small amount. This is the negative of the parameter typically referred to as tauC.
leapSeconds	Leap seconds introduced since 1980. Useful to convert to GPS time
comments	Miscellaneous comments found in the header of the RINEX file

For files using RINEX major version 3, it contains the following elements:

rinexVersion	Version of the RINEX format used in the file
rinexFileType	Type of RINEX file
satelliteSystem	Character indicating the satellite system. For GLONASS, it should be "R"
generatorProgram	Program used to generate the RINEX file
generatorEntity	Individual or organization that generated the file
fileCreationDateString	Date-time string indicating when the file was created

`systemTimeCorrectionType`
: String indicating the type of system time correction, which defines the exact meaning of the system time correction parameters A0 and A1 and between what time systems these allow conversion. Possible values are (the definition of A0/A1 are given between brackets): GAUT, for GAL to UTC (a0/a1); GPUT, for GPS to UTC (a0/a1); SBUT, for SBAS to UTC (a0/a1); GLUT, for GLO to UTC (TauC/0); GAGP or GPGA, for GPS to GAL (A0G/A1G); GLGP, for GLO to GPS (TauGPS/0); QZGP, for QZS to GPS (a0/a1); QZUT, for QZS to UTC (a0/a1); BDUT, for BDS to UTC (A0UTC/A1UTC); IRUT, for IRN to UTC (A0UTC/A1UTC); IRGP, for IRN to GPS (A0/A1). Note that GLONASS RINEX navigation files will typically contain system time corrections of types either GLUT or GLGP

`timeCorrectionA0`
: A0 parameter (bias) for system time correction in seconds. For corrections of type GLUT, this is equivalent to the field `sysTimeCorrection` found in headers of GLONASS RINEX navigation files of major version 2, i.e., the negative of tauC. For corrections of type GLGP, this is equivalent to the parameter tauGPS

`timeCorrectionA1`
: A1 parameter (drift) for system time correction in seconds/seconds. Note that this is set to 0 in GLUT and GLGP correction types

`timeCorrectionReferenceTime`
: Reference time for system time corrections, in seconds into GPS/GAL/BDS/QZS/IRN/SBAS week. The correction to be applied to the system time should be calculated as: Correction=A0 + A1*(ephemerisTime - timeCorrectionReferenceTime). Note that GLONASS time is aligned with UTC + 3 hours at a precision of 1 ms, and timeCorrectionReferenceTime is set to 0.

`timeCorrectionReferenceWeekNumber`
: Reference week number for the system time correction reference time. For GPS/GAL/QZS/IRN/SBAS, it is a continuous week scale since 6th of January 1980. For BDS, it is a continuous week scale since 1st of January 2006. For GLONASS, it is set to 0

`timeCorrectionSatelliteNumber`
: String indicating the GNSS satellite that is broadcasting the time system difference for system time correction

`UTCType`
: String indicating the exact UTC type (specific UTC realization) to which system time correction parameters refers to

`leapSeconds`
: Current number of leap seconds

`deltaTimeLeapSeconds`
: Number of leap seconds at a reference date for leap seconds specified by the following week number and week day. Note that the date specified by said week number and day can be either in the past or future

`deltaTimeLeapSecondsWeekNumber`
: Week number for the reference date to which deltaTimeLeapSeconds refers. Given as continuous week number from either 6th of January 1980 (GPS week number) or from 1st of January 2006 (BDS week number). Field leapSecondsTimeSystemIdentifier specifies which of the 2 systems is used.

`deltaTimeLeapSecondsDayNumber`
: Day of week number for the reference date to which deltaTimeLeapSeconds refers. Ranges from 1-7 for reference dates in GPS, or 0-6 for reference dates in BDS. The first day of each week (1 for GPS, 0 for BDS) is considered to be Sunday

`leapSecondsTimeSystemIdentifier`
: String indicating the time system used for the week number and day number for the future/past leap seconds reference date. Can be either "GPS" or "BDS" (BeiDou)

`leapSecondsTimeSystemIdentifier`
: String indicating the time system used for the week number and day number for the future/past leap seconds reference date. Can be either "GPS" or "BDS" (BeiDou)

`ionosphericCorrections`
: List where each element contains the fields required to perform a given type of ionospheric corrections. The contents of each element of this list are detailed below

`comments`
: Miscellaneous comments found in the header of the RINEX file

As mentioned above, for RINEX files of version 3.00 and above, the header element contains a list named `ionosphericCorrections`. Each element of this list is in turn a list itself, with the following elements:

`ionosphericCorrectionType`
: A string indicating the type of ionospheric correction for which this element provides coefficients. Can be GAL, GPSA, GPSB, QZSA, QZSB, BDSA, BDSB, IRNA or IRNB. The specific coefficients given in the following elements vary depending on the type of ionospheric correction. For GAL, 3 coefficients are given (a0-a2). For GPSA, QZSA, BDSA and IRNA, 4 coefficients are given (alpha0-alpha3). For GPSB, QZSB, BDSB and IRNB, 4 coefficients are given (beta0-beta1)

`coefficients`
: 3 (for GAL correction) or 4 (for all other corrections) elements providing the ionospheric correction coefficients. The names of the elements vary depending on the type of correction, as stated above

`ionosphericCorrectionTimeMark`
: Character indicating the transmission time. The field is only mandatory for corrections of type BDSA and BDSB. It can be a letter from "A" to "X", with each letter indicating a 1-hour interval: A=00h-01h, B=01h-02hB, ..., X=23h-24h

`ionosphericCorrectionTimeMark`
: Character indicating the transmission time. The field is only mandatory for corrections of type BDSA and BDSB. It can be a letter from "A" to "X", with each letter indicating a 1-hour interval: A=00h-01h, B=01h-02hB, ..., X=23h-24h

`ionosphericCorrectionSV`
: String indicating the satellite that provided the ionospheric correction parameters. The field is only mandatory for BDSA and BDSB corrections. If multiple sources are available for the same type of BDSA/BDSB correction, these should be given priority according to the satellite that provided them as follows: BDS GEO satellites first, followed by BDS IGSO satellites, and finally BDS MEO satellites

The second element is named `messages`, and it contains one element for each navigation message found in the RINEX file. Each of these elements is a list with the following elements that provide information about the position of the GLONASS satellite:

`satelliteNumber`
: Slot number of the satellite within the GLONASS constellation. It can be converted to a PRN code by adding 37 to it

`epochYear`
: Epoch year in 4-digit format.

`epochMonth`
: Epoch month

`epochDay`
: Epoch day

`epochHour`
: Epoch hour

`epochMinute`
: Epoch minute

`epochSecond`
: Epoch second

`ephemerisUTCTime`
: A nanotime object indicating the time corresponding to the reported position (ephemeris) in the present message. The time is in UTC, obtained by applying the individual clock bias of the particular satellite (`clockBias` field of each message) and the latest global GLONASS time bias with respect to UTC (`sysTimeCorrection` field of the header) to the uncorrected ephemeris time, given by the previous time fields. Corrections are performed as described in the GLONASS system specifications (http://gauss.gge.unb.ca/GLONASS.ICD.pdf)

`clockBias`
: Clock bias (i.e., constant offset) that should be applied to the satellite time in order to obtain an even more accurate UTC time. In seconds

`relativeFreqBias`
: Clock drift of the satellite clock that should be applied in combination with the time difference to the reference time in order to obtain an even more accurate UTC time. In seconds per second

`messageFrameTime`
: Second of the UTC day when the message was transmitted

`positionX`
: X coordinate of the position of the satellite in km, in the ITRF system of coordinates

`positionY`
: Y coordinate of the position of the satellite in km, in the ITRF system of coordinates

`positionZ`
: Z coordinate of the position of the satellite in km, in the ITRF system of coordinates

`velocityX`
: X component of the velocity of the satellite in km/s, in the ITRF system of coordinates

`velocityY`
: Y component of the velocity of the satellite in km/s, in the ITRF system of coordinates

`velocityZ`
: Z component of the velocity of the satellite in km/s, in the ITRF system of coordinates

`accelX`
: X component of the accel of the satellite in km/s, in the ITRF system of coordinates

`accelY` Y component of the accel of the satellite in km/s, in the ITRF system of coordinates

`accelZ` Z component of the accel of the satellite in km/s, in the ITRF system of coordinates

`satelliteHealthCode`
Code indicating the health of the satellite. 0 if healthy

`freqNumber` Frequency number (k) of the GLONASS satellite. The two frequencies in MHz, f1 and f2, used by the satellite to transmit data can be calculated as follows: f1 = 1602 + k*9/16 and f2 = 1246 + k*7/16

`informationAge` Age in days of the observation data used to generate the provided ephemeris

For GLONASS RINEX navigation files of version 3.05 and above, each message element contains the following additional elements:

`GLONASSType` String indicating the type of GLONASS satellite. Can be either "GLO" for fist-generation GLONASS satellites, or "GLO-M/K" for second or third generation satellites

`updatedDataFlag`
Logical indicating if the provided ephemeris data is up to date

`numberSatellitesAlmanac`
Number of satellites in the almanac for the current transmitted frame. Can be either 4 or 5. Note that almanac data provide coarser information about the location of multiple satellites in a GNSS constellation, and these are not actually included in RINEX navigation files.

`ephemerisValidityTimeInterval`
Length of the time interval for which the ephemeris is valid, in minutes. Can be 0, 30, 45 or 60.

`parityEphemerisValidityTimeInterval`
String indicating the evenness or oddity of the time interval for which ephemeris is valid. "Odd" for intervals of 45 minutes, and "Even" for the rest.

`tauCSource` String indicating the source providing the value of the tauC parameter, given in the header in field `sysTimeCorrection` for RINEX navigation messages of version 2, or field `timeCorrectionA0` for RINEX navigation messages of version 3 where a time system correction of type GLUT is given. Can be either "On-board" (if it was computed by the on-board satellite processor) or "Ground" (computed and uploaded to the satellite by control segment)

`tauGPSSource` String indicating the source providing the value of the tauGPS parameter, given in the header in field `timeCorrectionA0` for RINEX navigation messages of version 3 where a time system correction of type GLGP is given. Can be either "On-board" (if it was computed by the on-board satellite processor) or "Ground" (computed and uploaded to the satellite by control segment)

`totalGroupDelay`
Bias difference between codes broadcasted on L1 and the ionospheric-free combination of the codes broadcasted at L1 and L2, in seconds. This parameter, also known as timing group delay (TGD), should be considered when calculating satellite clock error.

URAI — Value of User Range Accuracy Index (URAI). This is an index giving a measurement of the accuracy of the GNSS ranging accuracy. It is an integer ranging from 0 to 15. 0 indicates the highest accuracy, corresponding to an accuracy of 1 m, while 14 indicates the lowest accuracy, of 512 m. A value of 15 indicates unknown accuracy. For a complete table of equivalence between URAI values and ranging accuracy in meters, see Table 4.4 of GLONASS Interface Control Document (http://gauss.gge.unb.ca/GLONASS.ICD.pdf)

almanacHealthStatus — String indicating the health status of the satellite provided in the almanac (as previously mentioned, full almanac data are not present in RINEX navigation files))

References

https://gage.upc.edu/gFD/ https://www.navcen.uscg.gov/pubs/gps/rinex/rinex.txt ftp://www.ngs.noaa.gov/cors/RINEX211.tx
http://acc.igs.org/misc/rinex304.pdf http://gauss.gge.unb.ca/GLONASS.ICD.pdf

Examples

```
# The file testGLONASSRINEXv2.txt provided with the package includes 5 navigation
# messages from 4 GLONASS satellites

testGLONASSnav <- readGLONASSNavigationRINEX(paste0(path.package("asteRisk"),
"/testGLONASSRINEXv2.txt"))
testGLONASSnav$header
testGLONASSnav$messages
```

---

readGPSNavigationRINEX

Read a RINEX navigation file for GPS satellites

Description

RINEX (Receiver Independent Exchange Format) is one of the most widely used formats for providing data of satellite navigation systems. The RINEX standard defines several structured text file types file types, among which navigation files are used to distribute positional information of the satellites. The exact information provided in a RINEX navigation file varies for each satellite navigation system. This function reads RINEX navigation files for satellites of the GPS constellation, operated by the USA.

Usage

```
readGPSNavigationRINEX(filename)
```

Arguments

filename — Path to the GPS RINEX navigation file.

Value

A list with two elements. The first element, named `header`, is a list with the information contained in the header of the RINEX file. For files using RINEX major version 2, it contains the following elements:

`rinexVersion` Version of the RINEX format used in the file

`rinexFileType` Type of RINEX file

`generatorProgram`
Program used to generate the RINEX file

`generatorEntity`
Individual or organization that generated the file

`fileCreationDateString`
Date-time string indicating when the file was created

`ionAlphaA0` Coefficient for ionospheric correction A0

`ionAlphaA1` Coefficient for ionospheric correction A1

`ionAlphaA2` Coefficient for ionospheric correction A2

`ionAlphaA3` Coefficient for ionospheric correction A3

`ionBetaB0` Coefficient for ionospheric correction B0

`ionBetaB1` Coefficient for ionospheric correction B1

`ionBetaB2` Coefficient for ionospheric correction B2

`ionBetaB3` Coefficient for ionospheric correction B3

`deltaUTCA0` A0 parameter, corresponding to bias between GPST and UTC time at the reference time (Tot) given by fields `referenceTimeUTC` and `referenceWeekUTC`. Should be used to compute accurate time in UTC

`deltaUTCA1` A1 parameter, corresponding to the clock drift between GPST and UTC at the reference time (Tot) given by fields `referenceTimeUTC` and `referenceWeekUTC`. Should be used to compute accurate time in UTC

`referenceTimeUTC`
Time in seconds of current UTC week of Tot, which is the reference time to correct GPST time to UTC

`referenceWeekUTC`
UTC reference week number (continuous scale, not modulo 1024) of Tot.

`leapSeconds` Leap seconds introduced since the 6th of January, 1980. Useful to convert to UTC time (UTC time = GPS time - leap seconds)

`comments` Miscellaneous comments found in the header of the RINEX file

For files using RINEX major version 3, it contains the following elements:

`rinexVersion` Version of the RINEX format used in the file

`rinexFileType` Type of RINEX file

`satelliteSystem`
Character indicating the satellite system. For GLONASS, it should be "R"

`generatorProgram`
Program used to generate the RINEX file

`generatorEntity`
: Individual or organization that generated the file

`fileCreationDateString`
: Date-time string indicating when the file was created

`systemTimeCorrectionType`
: String indicating the type of system time correction, which defines the exact meaning of the system time correction parameters A0 and A1 and between what time systems these allow conversion. Possible values are (the definition of A0/A1 are given between brackets): GAUT, for GAL to UTC (a0/a1); GPUT, for GPS to UTC (a0/a1); SBUT, for SBAS to UTC (a0/a1); GLUT, for GLO to UTC (TauC/0); GAGP or GPGA, for GPS to GAL (A0G/A1G); GLGP, for GLO to GPS (TauGPS/0); QZGP, for QZS to GPS (a0/a1); QZUT, for QZS to UTC (a0/a1); BDUT, for BDS to UTC (A0UTC/A1UTC); IRUT, for IRN to UTC (A0UTC/A1UTC); IRGP, for IRN to GPS (A0/A1). Note that GPS RINEX navigation files will typically contain system time corrections of type GPUT

`timeCorrectionA0`
: A0 parameter (bias) for system time correction in seconds

`timeCorrectionA1`
: A1 parameter (drift) for system time correction in seconds/seconds

`timeCorrectionReferenceTime`
: Reference time for system time corrections, in seconds into GPS/GAL/BDS/QZS/IRN/SBAS week. The correction to be applied to the system time should be calculated as: Correction=A0 + A1*(ephemerisTime - timeCorrectionReferenceTime)

`timeCorrectionReferenceWeekNumber`
: Reference week number for the system time correction reference time. For GPS/GAL/QZS/IRN/SBAS, it is a continuous week scale since 6th of January 1980. For BDS, it is a continuous week scale since 1st of January 2006. For GLONASS, it is set to 0

`timeCorrectionSatelliteNumber`
: String indicating the GNSS satellite that is broadcasting the time system difference for system time correction

`UTCType`
: String indicating the exact UTC type (specific UTC realization) to which system time correction parameters refers to

`leapSeconds`
: Current number of leap seconds

`deltaTimeLeapSeconds`
: Number of leap seconds at a reference date for leap seconds specified by the following week number and week day. Note that the date specified by said week number and day can be either in the past or future

`deltaTimeLeapSecondsWeekNumber`
: Week number for the reference date to which deltaTimeLeapSeconds refers. Given as continuous week number from either 6th of January 1980 (GPS week number) or from 1st of January 2006 (BDS week number). Field leapSecondsTimeSystemIdentifier specifies which of the 2 systems is used.

`deltaTimeLeapSecondsDayNumber`
: Day of week number for the reference date to which deltaTimeLeapSeconds refers. Ranges from 1-7 for reference dates in GPS, or 0-6 for reference dates

in BDS. The first day of each week (1 for GPS, 0 for BDS) is considered to be Sunday

`leapSecondsTimeSystemIdentifier`
: String indicating the time system used for the week number and day number for the future/past leap seconds reference date. Can be either "GPS" or "BDS" (BeiDou)

`leapSecondsTimeSystemIdentifier`
: String indicating the time system used for the week number and day number for the future/past leap seconds reference date. Can be either "GPS" or "BDS" (BeiDou)

`ionosphericCorrections`
: List where each element contains the fields required to perform a given type of ionospheric corrections. The contents of each element of this list are detailed below

`comments`
: Miscellaneous comments found in the header of the RINEX file

As mentioned above, for RINEX files of version 3.00 and above, the header element contains a list named `ionosphericCorrections`. Each element of this list is in turn a list itself, with the following elements:

`ionosphericCorrectionType`
: A string indicating the type of ionospheric correction for which this element provides coefficients. Can be GAL, GPSA, GPSB, QZSA, QZSB, BDSA, BDSB, IRNA or IRNB. The specific coefficients given in the following elements vary depending on the type of ionospheric correction. For GAL, 3 coefficients are given (a0-a2). For GPSA, QZSA, BDSA and IRNA, 4 coefficients are given (alpha0-alpha3). For GPSB, QZSB, BDSB and IRNB, 4 coefficients are given (beta0-beta1)

`coefficients`
: 3 (for GAL correction) or 4 (for all other corrections) elements providing the ionospheric correction coefficients. The names of the elements vary depending on the type of correction, as stated above

`ionosphericCorrectionTimeMark`
: Character indicating the transmission time. The field is only mandatory for corrections of type BDSA and BDSB. It can be a letter from "A" to "X", with each letter indicating a 1-hour interval: A=00h-01h, B=01h-02hB, ..., X=23h-24h

`ionosphericCorrectionTimeMark`
: Character indicating the transmission time. The field is only mandatory for corrections of type BDSA and BDSB. It can be a letter from "A" to "X", with each letter indicating a 1-hour interval: A=00h-01h, B=01h-02hB, ..., X=23h-24h

`ionosphericCorrectionSV`
: String indicating the satellite that provided the ionospheric correction parameters. The field is only mandatory for BDSA and BDSB corrections. If multiple sources are available for the same type of BDSA/BDSB correction, these should be given priority according to the satellite that provided them as follows: BDS GEO satellites first, followed by BDS IGSO satellites, and finally BDS MEO satellites

The second element is named `messages`, and it contains one element for each navigation message found in the RINEX file. Each of these elements is a list with the following elements that provide information about the position of the GPS satellite:

`satellitePRNCode`
: PRN code of the satellite. Unique PRN codes are assigned to all satellites in global navigation satellite systems, and therefore provide an identifier for each of them

`tocYear`
: Toc year in 4-digit format. Toc is the GPS time of the specific satellite that should be used as the time reference to apply clock bias, clock drift and possibly even clock drift rate, as well as a relativistic correction, as described in the GPS system specification (https://www.gps.gov/technical/icwg/IS-GPS-200H.pdf) to obtain the corrected GPST system time. The GPST system time can be converted to UTC time by subtracting leap seconds since the 6th of January 1980 and performing another polynomial correction to account for bias and drift between GPST and UTC times.

`tocMonth`
: Toc month

`tocDay`
: Toc day

`tocHour`
: Toc hour

`tocMinute`
: Toc minute

`tocSecond`
: Toc second

`clockBias`
: Clock bias (i.e., constant offset) at Toc that should be applied to the satellite time in order to calculate accurate GPST. In seconds. Often referred to as af0.

`clockDrift`
: Clock drift of the satellite clock at Toc that should be applied to the satellite time in order to calculate accurate GPST. In seconds. Often referred to as af1.

`clockDriftRate`
: Rate of change for the clock drift of the satellite clock at Toc. It is frequently 0, but if not, it should be applied in combination with clock bias and clock drift in order to correct to GPST as accurately as possible. In seconds per square second. Often referred to as af2.

`IODE`
: Issue-of-data ephemeris. It acts as a time-stamp or unique identifier for the provided navigation data. In particular, the IODE of a given navigation message should never be the same as the IODE for any other navigation message broadcasted by the same satellite in the past 6 days, although violations of this rule have been observed. Most frequently, IODE are not reused in a period of 7 days, so that they match exactly the IODC.

`radiusCorrectionSine`
: Amplitude of the sine harmonic component for the correction of orbital radius. In meters

`deltaN`
: Mean motion difference from computed value. In radians per second. In order to obtain the real (perturbed) mean motion, first the Keplerian mean motion should be calculated from the semi-major axis. Then, deltaN should be added to it.

`correctedMeanMotion`
: Corrected mean motion calculated by adding deltaN to the value computed from the semi-major axis. In radians per second

`meanAnomaly` Mean anomaly of the satellite at epoch. In radians. This indicates where the satellite is along its orbital path. It is provided as the angle between the direction of the perigee and the hypothetical point where the object would be if it was moving in a circular orbit with the same period as its true orbit after the same amount of time since it last crossed the perigee had ellapsed. Therefore, 0 denotes that the object is at the perigee. This is a Keplerian orbital element.

`latitudeCorrectionCosine`
Amplitude of the cosine harmonic component for the correction of latitude argument. In radians

`eccentricity` Eccentricity of the orbit of the satellite at epoch. This is a Keplerian orbital element.

`latitudeCorrectionSine`
Amplitude of the sine harmonic component for the correction of latitude argument. In radians

`semiMajorAxis` Semi-major axis of the orbit of the satellite at epoch. In meters. This is a Keplerian orbital element

`toeSecondsOfGPSWeek`
Time of the GPS week (in seconds) for the ephemeris. Together with the `toeGPSWeek`, it can be used to calculate the ephemeris time in GPS time of the specific satellite, to which sequential corrections first to GPST and then to UTC should be applied.

`inclinationCorrectionCosine`
Amplitude of the cosine harmonic component for the correction of inclination. In radians

`ascension` Longitude of the ascending node of the satellite at epoch, also known as right ascension of the ascending node, in radians. This is the angle between the direction of the ascending node (the point where the satellite crosses the equatorial plane moving north) and the direction of the First Point of Aries (which indicates the location of the vernal equinox). This is a Keplerian orbital element.

`inclinationCorrectionSine`
Amplitude of the sine harmonic component for the correction of inclination. In radians

`inclination` Mean orbital inclination of the satellite in radians. This is the angle between the orbital plane of the satellite and the equatorial plane. This is a Keplerian orbital element.

`radiusCorrectionCosine`
Amplitude of the cosine harmonic component for the correction of orbital radius. In meters.

`perigeeArgument`
Mean argument of the perigee of the object in radians. This is the angle between the direction of the ascending node and the direction of the perigee (the point of the orbit at which the object is closest to the Earth). This is a Keplerian orbital element.

`OMEGADot` Angular velocity of the satellite with respect to the vernal equinox. In radians/second.

`codesL2Channel` Flag indicating if coarse/acquisition (C/A) code is being transmitted on the L2 channel (value of 1) or not (value of 0)

`toeGPSWeek` GPS week number at epoch

`dataFlagL2P` Flag indicating if precise (P) code is being transmitted on the L2 channel (value of 1) or not (value of 0)

`satelliteAccuracy`
Accuracy of the position of the satellite, in meters.

`satelliteHealthCode`
Code indicating the health of the satellite. 0 if healthy.

`totalGroupDelay`
Bias difference between codes broadcasted on L1 and the ionospheric-free combination of the codes broadcasted at L1 and L2, in seconds. This parameter, also known as timing group delay (TGD), should be considered when calculating satellite clock error.

`IODC` Issue-of-data clock. It acts as a time-stamp or unique identifier for the provided navigation data. In particular, the IODC of a given navigation message should never be the same as the IODC for any other navigation message broadcasted by the same satellite in the past 7 days, although violations of this rule have been observed. Most frequently, IODE are not reused in a period of 7 days instead of the mandatory 6 days, so that they match exactly the IODC.

`transmissionTime`
Transmission time for the navigation message, in seconds of GPS week.

`fitInterval` Flag indicating for how long the broadcasted ephemeris are valid since the last time the data was updated. It should be noted that in order to obtain positional values/orbital elements at times other than epoch, the corrections for perturbed orbital elements should be applied and propagated. If 0, the ephemeris data are valid for up to 4 hours. If 1, they are valid for more than 4 hours.

`ephemerisUTCTime`
A nanotime object indicating the time corresponding to the reported position (ephemeris) in the present message. The time is in UTC, obtained by first applying the individual clock bias, clock drift and clock drift rate of the particular satellite (fields `clockBias`, `clockDrift` and `clockDriftRate` of each message) and a relativistic correction to obtain corrected GPST time (system-wide GPS time), and then subtraction of leap seconds since the 6th of January 1980 and a second polynomial correction (with fields `deltaUTCA0` and `deltaUTCA1` from the header) to obtain UTC time. Corrections are performed as described in the GPS system specifications (https://www.gps.gov/technical/icwg/IS-GPS-200H.pdf).

`position_ITRF` Position of the satellite in the ITRF frame, calculated from the provided orbital ephemeris following the procedure described in the GPS system specifications. In meters.

`velocity_ITRF` Velocity of the satellite in the ITRF frame, calculated from the provided orbital ephemeris following the procedure described in the GPS system specifications. In meters/second.

`acceleration_ITRF`
Acceleration of the satellite in the ITRF frame, calculated from the provided orbital ephemeris following the procedure described in the GPS system specifications. In meters/squared second.

References

https://gage.upc.edu/gFD/ https://www.navcen.uscg.gov/pubs/gps/rinex/rinex.txt ftp://www.ngs.noaa.gov/cors/RINEX211.tx
http://acc.igs.org/misc/rinex304.pdf https://www.icao.int/Meetings/AMC/MA/2004/GNSS/icd.pdf https://www.gps.gov/tech
GPS-200H.pdf

Examples

```
# The file testGPSRINEXv2.txt provided with the package includes 3 navigation
# messages from 3 GPS satellites

testGPSnav <- readGPSNavigationRINEX(paste0(path.package("asteRisk"),
"/testGPSRINEXv2.txt"))
testGPSnav$header
testGPSnav$messages
```

---

readOEM *Read an Orbital Ephemeris Message file*

Description

OEM (Orbital Ephemeris Message) is one of the three standard file formats defined by the CCSDS for transferring spacecraft orbit information. OEM files contain the position and velocity of a given object at multiple times (epochs). They can also contain optionally acceleration values, covariance matrixes that indicate the uncertainty of the provided state vectors and other additional information. This function reads OEM files, retrieving also the optional fields.

Usage

```
readOEM(filename)
```

Arguments

`filename`	Path to the OEM file.

Value

A list with two elements. The first element, named `header`, is a list with the following elements:

`OMVersion`	Version of the OEM format used in the file
`creationDate`	Date of creation of the file
`creator`	Individual or organization that generated the file

The second element is named `dataBlocks`, and it contains one element for each ephemeris data block found in the OEM file. Each of these elements is a list with the following elements that provide information about the ephemerides of object (note that some elements are not mandatory and therefore might not be present in all OEM files; in these cases, their value is set to NULL):

`objectName`	Name of the object

`objectID` Object identifier for the object. Frequently, although not always, the identifier has the format YYYY-NNNP(PP), where YYYY is the year of launch, NNN is the three-digit serial number specifying the launch number during year YYYY and P(PP) is a part specifier comprising 1 to 3 capital letters that indicate the part of the object put into space during the launch.

`referenceFrame` Frame of reference in which ephemerides are provided.

`refFrameEpoch` Epoch for the frame of reference, for cases where it is not intrinsic to the frame itself, such as TEME.

`centerName` Name of the center of coordinates. For example, a celestial body, the barycenter of the entire Solar System or even other spacecraft.

`timeSystem` Time system used for the ephemerides, covariance matrixes and all other time fields of the data block.

`startTime` Start time of the time span covered by the ephemerides and covariance matrixes in this data block.

`endTime` End time of the time span covered by the ephemerides and covariance matrixes in this data block.

`usableStartTime`
Start time of the usable time span covered by the ephemerides and covariance matrixes in this data block.

`usableEndTime` End time of the usable time span covered by the ephemerides and covariance matrixes in this data block.

`interpolationMethod`
Recommended interpolation method to calculate ephemerides at epochs between those directly given in the data block.

`interpolationOrder`
Recommended interpolation degree to calculate ephemerides at epochs between those directly given in the data block.

`mass` Mass in kg of the object.

`dragArea` Effective area of the object subjected to drag, in square meters.

`dragCoefficient`
Drag coefficient of the object.

`solarRadArea` Effective area of the object subjected to solar radiation pressure, in square meters.

`solarRadCoefficient`
Solar radiation pressure coefficient of the object.

`ephemerides` Data frame with the 7 or 10 columns providing the ephemerides for the object. The 1st column provides the epochs for each ephemeris; columns 2 to 4 provide the X, Y and Z components of position (in km), and columns 5 to 7 provide the X, Y and Z components of velocity (in km/s). Columns 8 to 10 are optional, and if present provide the X, Y and Z components of acceleration (in km/s2)

`covarianceMatrixes`
List where each element is a 3-element list that provides a covariance matrix for this data block. Each of the 3-element lists corresponding to a covariance matrix contains the following elements:

epoch Epoch of the navigation solution related to this covariance matrix.

referenceFrame Reference frame for the covariance matrix. Frequently this is the same as for the ephemerides. In order to facilitate interpretation of the covariance matrix, conversion to a perifocal frame of reference might be advisable.

covarianceMatrix Covariance matrix that provides information about the uncertainties of position and velocities. This is a symmetric 6x6 matrix where the values in the diagonal are the squared standard deviations of each variable, and the other values are covariances between 2 variables (those corresponding to the row and column of each value). The rows and columns correspond to the following variables, in the specified order: X position, Y position, Z position, X velocity, Y velocity and Z velocity.

References

https://public.ccsds.org/Pubs/502x0b2c1e2.pdf https://spotthestation.nasa.gov/trajectory_data.cfm

Examples

```
# The file testOEM.txt provided with the package includes ephemerides data
# for the ISS publicly available

testOEM_ISS <- readOEM(paste0(path.package("asteRisk"), "/testOEM.txt"))
testOEM_ISS$header
testOEM_ISS$dataBlocks[[1]]$objectName
head(testOEM_ISS$dataBlocks[[1]]$ephemerides)
```

---

readTLE *Read a TLE file*

Description

TLE (Two-/Three- Line Element) is a standard structured text file format for representing orbital state vectors. This function reads a TLE file containing one or more TLEs. The TLE file can contain either Two Line Elements or Three Line Elements, but all the TLE in a single file must be of the same type. The two lines of a Two Line Element contain all the ephemeris information. The additional line in a Three Line Element is optional, and contains just the satellite name. For a detailed description of the TLE format, see https://celestrak.com/columns/v04n03/#FAQ01.

Usage

```
readTLE(filename, maxTLEs=NULL)
```

Arguments

filename	Path to the TLE file. Alternatively, an URL pointing to a TLE file.
maxTLEs	Maximum number of TLEs to read, starting from the beginning of the file. By default, all TLEs present in the file are read.

Value

A list with the following elements that define the orbital state vector of the satellite (or, if the file contained multiple TLE, a nested list, where each element of the top level list represents an orbital state vector, and comprises the following elements):

`NORADcatalogNumber`
: NORAD Catalog Number, also known as Satellite Catalog Number, assigned by United States Space Command to each artificial object orbiting Earth

`classificationLevel`
: Classification level of the information for the orbiting object. Can be unclassified, classified, secret or unknown

`internationalDesignator`
: International Designator, also known as COSPAR ID, of the object. It consists of the launch year, separated by a hyphen from a three-digit number indicating the launch number for that year and a set of one to three letters indicating the piece for a launch with multiple pieces.

`launchYear`
: The launch year of the object

`launchNumber`
: The launch number of the object during its launch year

`launchPiece`
: The piece for the launch of the object, if it was a launch with multiple pieces

`dateTime`
: Date time string to which the orbital state vector corresponds

`elementNumber`
: Element number for the object. In principle, every time a new TLE is generated for an object, the element number is incremented, and therefore element numbers could be used to assess if all the TLEs for a certain object are available. However, in practice it is observed that this is not always the case, with some numbers skipped and some numbers repeated.

`inclination`
: Mean orbital inclination of the satellite in degrees. This is the angle between the orbital plane of the satellite and the equatorial plane

`ascension`
: Mean longitude of the ascending node of the satellite at epoch, also known as right ascension of the ascending node, in degrees. This is the angle between the direction of the ascending node (the point where the satellite crosses the equatorial plane moving north) and the direction of the First Point of Aries (which indicates the location of the vernal equinox)

`eccentricity`
: Mean eccentricity of the orbit of the object. Eccentricity is a measurement of how much the orbit deviates from a circular shape, with 0 indicating a perfectly circular orbit and 1 indicating an extreme case of parabolic trajectory

`perigeeArgument`
: Mean argument of the perigee of the object in degrees. This is the angle between the direction of the ascending node and the direction of the perigee (the point of the orbit at which the object is closest to the Earth)

`meanAnomaly`
: Mean anomaly of the orbit of the object in degrees. This indicates where the satellite is along its orbital path. It is provided as the angle between the direction of the perigee and the hypothetical point where the object would be if it was moving in a circular orbit with the same period as its true orbit after the same amount of time since it last crossed the perigee had ellapsed. Therefore, 0 denotes that the object is at the perigee

`meanMotion` Mean motion of the satellite at epoch in revolutions/day

`meanMotionDerivative`
First time derivative of the mean motion of the satellite in revolutions/day^2^

`meanMotionSecondDerivative`
Second time derivative of the mean motion of the satellite in revolutions/day^3^.

`Bstar` Drag coefficient of the satellite in units of (earth radii)^-1^. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.

`ephemerisType` Source for the ephemeris (orbital state vector). Most commonly, it is distributed data obtained by combaining multiple observations with the SGP4/SDP4 models

`epochRevolutionNumber`
Number of full orbital revolutions completed by the object

`objectName` Name of the object, retrieved from the first line of the TLE if a Three Line Element was provided

References

https://celestrak.org/columns/v04n03/#FAQ01 http://www.celestrak.org/publications/aiaa/2006-6753/AIAA-2006-6753.pdf

Examples

```
# The file testTLE.txt provided with the package includes 29 TLE covering a
# variety of satellites, extracted from Revisiting Space Track Report #3

test_TLEs <- readTLE(paste0(path.package("asteRisk"), "/testTLE.txt"))
test_TLEs
```

---

registerPlanetaryEphemerides

Register a Planetary Ephemerides File

Description

This function registers a binary SPICE ephemerides file (with .bsp extension) as a JPL DE ephemerides file. The file is copied into a sub directory of the user data directory for asteRisk, and a message is displayed to confirm the registration.

Usage

```
registerPlanetaryEphemerides(ephemeridesFile)
```

Arguments

`ephemeridesFile`
The path to the ephemerides file to be registered. The file have a .bsp extension, and be a JPL DE ephemerides file.

Value

Returns TRUE if the file is successfully registered.

References

NAIF/SPICE Documentation: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/spk.html

Examples

```
## Not run:
# Assuming "de430.bsp" is a valid ephemerides file in the current directory
registerPlanetaryEphemerides("de430.bsp")

## End(Not run)
```

`revDay2radMin` *Converts revolutions per day to radians per minute*

Description

This function converts a rotation rate in revolutions per day to radians per minute. This conversion is useful since values in TLEs are given in revolutions per day, but the SGP4 and SDP4 propagators require the mean motion to be provided in radians per minute.

Usage

```
revDay2radMin(revPerDay)
```

Arguments

`revPerDay`	Value of the rotation rate in revolutions per day.

Value

The corresponding value of the rotation rate in radians per minute.

Examples

```
revDay2radMin(2)
```

sdp4 *Propagate an orbital state vector with the SDP4 model*

Description

Given an orbital state vector of a satellite, applies the SDP4 model to propagate its orbit to the desired time point. This allows the calculation of the position and velocity of the satellite at different times, both before and after the time corresponding to the known state vector (referred to as "epoch"). Kepler's equation is solved through fixed-point integration. The SDP4 model is a modified version of the SGP4 model that takes into account the secular and periodic perturbations of the Moon and the Sun on the orbit of the satellite. It also considers the Earth resonance effects on 24-hour geostationary and 12-hour Molniya orbits. Thanks to this, the SDP4 model can correctly propagate the orbit of objects in deep space (with orbital periods larger than 225 minutes, corresponding to altitudes higher than 5877.5 km). However, it should be noted that SDP4 employs only simplified drag equations, and lacks corrections for low-perigee orbits. Therefore, it is recommended to apply the standard SGP4 model (available through the sgp4 function) for satellites that are not in deep space. This implementation is based on a previous SDP4 implementation in Julia (SatelliteToolbox).

Usage

```
sdp4(n0, e0, i0, M0, omega0, OMEGA0, Bstar, initialDateTime, targetTime,
     keplerAccuracy=10e-12, maxKeplerIterations=10)
```

Arguments

n0	Mean motion of the satellite at epoch in radians/min.
e0	Mean eccentricity of the orbit of the satellite at epoch. Eccentricity ranges from 0 (perfectly circular orbit) to 1 (parabolic trajectory).
i0	Mean orbital inclination of the satellite at epoch in radians.
M0	Mean anomaly of the satellite at epoch.
omega0	Mean argument of perigee of the satellite at epoch.
OMEGA0	Mean longitude of the ascending node of the satellite at epoch. Also known as right ascension of the ascending node.
Bstar	Drag coefficient of the satellite in units of (earth radii)^-1^. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.
initialDateTime	Date-time string in UTC indicating the time corresponding to the known state vector of the satellite. Unlike for SGP4, it must be provided in all cases since it is required to calculate Moon and Sun perturbations.
targetTime	Time at which the position and velocity of the satellite should be calculated. It can be provided in two different ways: either as a number corresponding to the time in minutes counting from epoch at which the orbit should be propagated, or as a date-time string in UTC.

`keplerAccuracy` Accuracy to consider Kepler´s equation solved. If two consecutive solutions differ by a value lower than this accuracy, integration is considered to have converged.

`maxKeplerIterations` Maximum number of iterations after which fixed-point integration of Kepler's equation will stop, even if convergence according to the accuracy criterion has not been reached.

Value

A list with three elements. The first two elements represent the position and velocity of the satellite at the target time, in the TEME (True Equator, Mean Equinox) frame of reference. Position values are in km, and velocity values are in km/s. Each of these two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order. The third element indicates the algorithm used to propagate the orbit (sdp4).

References

https://juliapackages.com/p/satellitetoolbox https://celestrak.org/NORAD/documentation/spacetrk.pdf
http://www.celestrak.org/publications/aiaa/2006-6753/AIAA-2006-6753.pdf

Examples

```
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Calculation of the orbital period

2*pi/n0

# The period is higher than 225 min, and therefore the SDP4 model should be
# applied. Furthermore, from the mean motion in revolutions/day, it can be
# seen that it is a geostarionary satellite with a 24-hour period. Let´s
# calculate and compare the position and velocity of the satellite at epoch
# and 1 day later.

state_0 <- sdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                Bstar=Bstar, initialDateTime=epochDateTime, targetTime=0)
state_1day <- sdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                   Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)
state_0
state_1day
```

```
# The position and velocity are very similar after a full day, in accordance
# with the geostationary orbit
```

---

sgdp4 *Propagate an orbital state vector with the SGP4/SDP4 model*

Description

Given an orbital state vector of a satellite, applies the SGP4 or SDP4 model to propagate its orbit to the desired time point, as appropriate depending on the orbital period. The model will be automatically chosen depending on the orbital period. For objects in deep space (with orbital periods larger than 225 minutes, equivalent to altitudes higher than 5877.5 km) the SDP4 model will be applied. For objects near Earth (orbital periods shorter than 225 minutes, or altitudes lower than 5877.5 km) the SGP4 model will be used. It is not recommended to apply SGP4 to objects in deep space or SDP4 to objects near Earth, but this can be forced by calling directly the sgp4 and sdp4 functions.

Usage

```
sgdp4(n0, e0, i0, M0, omega0, OMEGA0, Bstar, initialDateTime=NULL, targetTime,
      keplerAccuracy=10e-12, maxKeplerIterations=10)
```

Arguments

n0	Mean motion of the satellite at epoch in radians/min.
e0	Mean eccentricity of the orbit of the satellite at epoch. Eccentricity ranges from 0 (perfectly circular orbit) to 1 (parabolic trajectory).
i0	Mean orbital inclination of the satellite at epoch in radians.
M0	Mean anomaly of the satellite at epoch.
omega0	Mean argument of perigee of the satellite at epoch.
OMEGA0	Mean longitude of the ascending node of the satellite at epoch. Also known as right ascension of the ascending node.
Bstar	Drag coefficient of the satellite in units of (earth radii)^-1^. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.
initialDateTime	Date-time string in UTC indicating the time corresponding to the known state vector of the satellite. It must be provided for objects in deep space, and also for objects near Earth if `targetTime` is provided as a date-time string.
targetTime	Time at which the position and velocity of the satellite should be calculated. It can be provided in two different ways: either as a number corresponding to the time in minutes counting from epoch at which the orbit should be propagated, or as a date-time string in UTC, in which case the date-time string for epoch must be provided through `initialDateTime`.

keplerAccuracy Accuracy to consider Kepler´s equation solved. If two consecutive solutions differ by a value lower than this accuracy, integration is considered to have converged.

maxKeplerIterations
Maximum number of iterations after which fixed-point integration of Kepler's equation will stop, even if convergence according to the accuracy criterion has not been reached.

Value

A list with three elements. The first two elements represent the position and velocity of the satellite at the target time, in the TEME (True Equator, Mean Equinox) frame of reference. Position values are in km, and velocity values are in km/s. Each of these two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order. The third element indicates the algorithm used to propagate the orbit (sgp4 or sdp4).

References

https://celestrak.org/NORAD/documentation/spacetrk.pdf http://www.celestrak.org/publications/aiaa/2006-6753/AIAA-2006-6753.pdf

Examples

```
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Calculation of the orbital period

2*pi/n0

# The period is higher than 225 min, and therefore the SDP4 model should be
# applied. Let´s calculatethe position and velocity of the satellite 12 hours
# after epoch.

italsat_12h <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                     Bstar=Bstar, initialDateTime=epochDateTime, targetTime=0)
italsat_12h$algorithm

# The SDP4 model was correctly chosen.
```

sgp4 *Propagate an orbital state vector with the SGP4 model*

Description

Given an orbital state vector of a satellite, applies the SGP4 model to propagate its orbit to the desired time point. This allows the calculation of the position and velocity of the satellite at different times, both before and after the time corresponding to the known state vector (referred to as "epoch"). Kepler's equation is solved through fixed-point integration. The SGP4 model can only accurately propagate the orbit of objects near Earth (with an orbital period shorter than 225 minutes, corresponding approximately to an altitude lower than 5877.5 km). For propagation of objects in deep space, the SDP4 model should be used, available through the `sdp4` function. This implementation is based on the theory and implementation described in Space Track Report #3, and includes the corrections summarized in Revisiting Space Track Report #3.

Usage

```
sgp4(n0, e0, i0, M0, omega0, OMEGA0, Bstar, initialDateTime=NULL, targetTime,
     keplerAccuracy=10e-12, maxKeplerIterations=10)
```

Arguments

`n0`	Mean motion of the satellite at epoch in radians/min.
`e0`	Mean eccentricity of the orbit of the satellite at epoch. Eccentricity ranges from 0 (perfectly circular orbit) to 1 (parabolic trajectory).
`i0`	Mean orbital inclination of the satellite at epoch in radians.
`M0`	Mean anomaly of the satellite at epoch.
`omega0`	Mean argument of perigee of the satellite at epoch.
`OMEGA0`	Mean longitude of the ascending node of the satellite at epoch. Also known as right ascension of the ascending node.
`Bstar`	Drag coefficient of the satellite in units of (earth radii)^-1^. Bstar is an adjusted value of the ballistic coefficient of the satellite, and it indicates how susceptible it is to atmospheric drag.
`initialDateTime`	Optional date-time string in UTC indicating the time corresponding to the known state vector of the satellite. It must be provided if `targetTime` is provided as a date-time string.
`targetTime`	Time at which the position and velocity of the satellite should be calculated. It can be provided in two different ways: either as a number corresponding to the time in minutes counting from epoch at which the orbit should be propagated, or as a date-time string in UTC, in which case the date-time string for epoch must be provided through `initialDateTime`.
`keplerAccuracy`	Accuracy to consider Kepler´s equation solved. If two consecutive solutions differ by a value lower than this accuracy, integration is considered to have converged.

maxKeplerIterations
: Maximum number of iterations after which fixed-point integration of Kepler's equation will stop, even if convergence according to the accuracy criterion has not been reached.

Value

A list with three elements. The first two elements represent the position and velocity of the satellite at the target time, in the TEME (True Equator, Mean Equinox) frame of reference. Position values are in km, and velocity values are in km/s. Each of these two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order. The third element indicates the algorithm used to propagate the orbit (sgp4).

References

https://celestrak.org/NORAD/documentation/spacetrk.pdf http://www.celestrak.org/publications/aiaa/2006-6753/AIAA-2006-6753.pdf

Examples

```
# The following orbital parameters correspond to an object with NORAD catalogue
# number 88888 the 1st of October, 1980 at 23:41:24 UTC.

n0 <- 16.05824518*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.0086731 # mean eccentricity at epoch
i0 <- 72.8435*pi/180 # mean inclination at epoch in radians
M0 <- 110.5714*pi/180 # mean anomaly at epoch in radians
omega0 <- 52.6988*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 115.9689*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 0.66816e-4 # drag coefficient

# Calculation of the orbital period

2*pi/n0

# The period is lower than 225 min, and therefore the SGP4 model is valid.
# Let´s calculate the position and velocity of the satellite 40 minutes after
# epoch

new_state <- sgp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                  Bstar=Bstar,targetTime = 40)
new_state
```

TEMEtoGCRF *Convert coordinates from TEME to GCRF*

Description

The TEME (True Equator, Mean Equinox) and GCRF (Geocentric Celestial Reference Frame) are both ECI frames of reference, i.e., Earth-centered inertial coordinate frames, where the origin is placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation).

The difference between the two resides in the fact that in the GCRF frame, the X-axis and Z-axis are aligned respectively with the mean equinox and rotation axis of Earth at 12:00 Terrestrial Time on the 1st of January, 2000, while in the TEME frame they are aligned with the mean equinox and rotation axis at the time of the corresponding TLE. Due to the change of the direction of the vernal equinox and the rotation axis over time, coordinates in the two frames differ slightly.

The function implement 2 algorithms. The default one applies first a transformation to ITRF using the algorithm implemented by David A. Vallado in the teme2ecef routine, followed by a transformation to GCRF using nutation-precesion-bias and polar motion matrices (transformation for velocity takes into account variations in Earth's rotation speed through length of day data provided by IERS). The second one follows the same procedure implemented by NAIF in the SPICE routine ZZTEME, and is provided for cases where users aim to reproduce SPICE's output.

This function requires the `asteRiskData` package, which can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
TEMEtoGCRF(position_TEME, velocity_TEME, dateTime, SPICEAlgorithm, ephemerisTime)
```

Arguments

`position_TEME`	Vector with the X, Y and Z components of the position of an object in TEME frame, in m.
`velocity_TEME`	Vector with the X, Y and Z components of the velocity of an object in TEME frame, in m/s.
`dateTime`	Date-time string with the date and time in UTC corresponding to the provided position and velocity vectors. This specifies the time for which the conversion from TEME to GCRF coordinates will be performed. It is required due to the change in the exact position of the rotation axis of Earth due to precesion, nutation and polar motion. Either `dateTime` or `ephemerisTime` must be provided.
`SPICEAlgorithm`	Logical indicating if the algorithm implemented in SPICE's ZZTEME routine should be used instead of the default algorithm. By default, `SPICEAlgorithm=FALSE`, and SPICE's algorithm is not applied.
`ephemerisTime`	Time in TDB seconds since J2000 for which the conversion from TEME to GCRF coordinates will be performed. This is also known as "ephemeris time" in SPICE. Either `dateTime` or `ephemerisTime` must be provided.

Value

A list with two elements representing the position and velocity of the satellite in the ECEF (Earth Centered, Earth Fixed) frame of reference. Position values are in m, and velocity values are in m/s. Each of the two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order.

References

https://celestrak.org/columns/v04n03/#FAQ01

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Let´s calculate the position and velocity of the satellite 1 day later

state_1day_TEME <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                         Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)

# We can now convert the results in TEME frame to GCRF frame, previously
# multiplying by 1000 to convert the km output of sgdp4 to m

state_1day_GCRF <- TEMEtoGCRF(state_1day_TEME$position*1000,
                              state_1day_TEME$velocity*1000,
                              "2006-06-27 00:58:29.34")
}
```

`TEMEtoITRF` *Convert coordinates from TEME to ITRF*

Description

The TEME (True Equator, Mean Equinox) frame of reference is an Earth-centered inertial coordinate frame, where the origin is placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The coordinates and velocities calculated with the SGP4 and SDP4 models are in the TEME frame of reference. This function converts positions and velocities in TEME to the ITRF (International Terrestrial Reference Frame), which is an ECEF (Earth Centered, Earth Fixed) frame of reference. In the ITRF, the origin is also placed at the center of mass of Earth, but the frame rotates with respect to the stars to remain fixed with respect to the Earth surface as it rotates. The Z-axis extends along the true North as defined by the IERS reference pole, and the X-axis extends towards the intersection between the equator and the Greenwich meridian at any time.

This function requires the `asteRiskData` package, which can be installed by running `install.packages('asteRiskData', repos='https://rafael-ayala.github.io/drat/')`

Usage

```
TEMEtoITRF(position_TEME, velocity_TEME, dateTime)
```

Arguments

position_TEME	Vector with the X, Y and Z components of the position of an object in TEME frame, in m.
velocity_TEME	Vector with the X, Y and Z components of the velocity of an object in TEME frame, in m/s.
dateTime	Date-time string with the date and time in UTC corresponding to the provided position and velocity vectors. This specifies the time for which the conversion from TEME to ITRF coordinates will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of TEME coordinates refers varies with time due to the motion of Earth.

Value

A list with two elements representing the position and velocity of the satellite in the ITRF (International Terrestrial Reference Frame) frame of reference. Position values are in m, and velocity values are in m/s. Each of the two elements contains three values, corresponding to the X, Y and Z components of position and velocity in this order.

References

https://celestrak.org/columns/v04n03/#FAQ01

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Let´s calculate the position and velocity of the satellite 1 day later

state_1day_TEME <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                         Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)

# We can now convert the results in TEME frame to ITRF frame, previously
# multiplying by 1000 to convert the km output of sgdp4 to m

state_1day_ITRF <- TEMEtoITRF(state_1day_TEME$position*1000,
```

```
                              state_1day_TEME$velocity*1000,
                              "2006-06-27 00:58:29.34")
}
```

`TEMEtoLATLON` *Convert coordinates from TEME to geodetic latitude, longitude and altitude*

Description

The TEME (True Equator, Mean Equinox) frame of reference is an Earth-centered inertial coordinate frame, where the origin is placed at the center of mass of Earth and the coordinate frame is fixed with respect to the stars (and therefore not fixed with respect to the Earth surface in its rotation). The coordinates and velocities calculated with the SGP4 and SDP4 models are in the TEME frame of reference. This function converts position in TEME to geodetic latitude, longitude and altitude, which can be considered to be a non-inertial, Earth-centered frame of reference.

Usage

```
TEMEtoLATLON(position_TEME, dateTime, degreesOutput=TRUE)
```

Arguments

`position_TEME`	Vector with the X, Y and Z components of the position of an object in TEME frame, in m.
`dateTime`	Date-time string with the date and time in UTC corresponding to the provided position vector. This specifies the time for which the conversion from TEME to geodetic coordinates will be performed. It is important to provide an accurate value, since the point over the surface of Earth to which a set of TEME coordinates refers varies with time due to the motion of Earth.
`degreesOutput`	Logical indicating if the output should be in sexagesimal degrees. If `degreesOutput=FALSE`, the output will be in radians.

Value

A vector with three elements, corresponding to the latitude and longitude in degrees (or radians if specified) and the altitude in m.

References

https://arc.aiaa.org/doi/10.2514/6.2006-6753

Examples

```
if(requireNamespace("asteRiskData", quietly = TRUE)) {
# The following orbital parameters correspond to an object with NORAD catalogue
# number 24208 (Italsat 2) the 26th of June, 2006 at 00:58:29.34 UTC.

n0 <- 1.007781*((2*pi)/(1440))  # Multiplication by 2pi/1440 to convert to radians/min
e0 <- 0.002664 # mean eccentricity at epoch
i0 <- 3.8536*pi/180 # mean inclination at epoch in radians
M0 <- 48.3*pi/180 # mean anomaly at epoch in radians
omega0 <- 311.0977*pi/180 # mean argument of perigee at epoch in radians
OMEGA0 <- 80.0121*pi/180 # mean longitude of ascending node at epoch in radians
Bstar <- 1e-04 # drag coefficient
epochDateTime <- "2006-06-26 00:58:29.34"

# Let´s calculate the position and velocity of the satellite 1 day later

state_1day_TEME <- sgdp4(n0=n0, e0=e0, i0=i0, M0=M0, omega0=omega0, OMEGA0=OMEGA0,
                         Bstar=Bstar, initialDateTime=epochDateTime, targetTime=1440)

# We can now convert the results in TEME frame to geodetic latitude, longitude
# and altitude, previously multiplying by 1000 to convert the km output of
# sgdp4 to m

state_1day_geodetic <- TEMEtoLATLON(state_1day_TEME$position*1000,
                                    "2006-06-27 00:58:29.34")
}
```

Index