ATTITUDE PACAKGE MANUAL

ATTITUDE is a Julia package for the orbital-attitude dynamical propagation. All the code is contained in the source folder src. This package only contains functions. Examples of main scripts using ATTITUDE are contained in the folder workspace.

In particular, besides other functions, ATTITUDE contains an osculating and a semi-analytical propagator for the coupled attitude-orbital dynamics. For the propagator for the full dynamics: see Propagators_girf.jl. For the semi-analytical propagator, see Semianalyticalpropagators_girf.jl

IMPORTANT:

The osculating and semi-analytical propagators works properly only if the satellites geometry and properties and the attitude initial conditions are given with respect to a rotating frame centred in the satellite's centre of mass, in principal axes, such that the principal moments of inertia are

A<=B<=C

With

A = int y^2+z^2 dm

B = int x^2+z^2 dm

C = int y^2+x^2 dm

See sections 15 and 22

CONTENTS

1. EXTERNAL DEPENDENCIES

The ATTITUDE package depends on the following pre-existing packages and projects:

- LinearAlgebra
- Elliptic
- DifferentialEquations
- DataInterpolations
- SatelliteToolboxAtmosphericModels
- SatelliteToolboxTransformations
- Polynomials
- SpecialFunctions
- HypergeometricFunctions
- FastGaussQuadrature
- Plots

The package Plots is introduced only for potential debugging purposes.

2. ATTITUDE.JL

This is the module of the package, used to organize the code of the package and specify the dependencies on

3. FUNCTIONSLIBRARY.JL

This file is a library of several generic functions, used in the package.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: -

3.1 MYFIX

This function rounds the input real number towards zero

INPUTS:	**numb** = real number
OUTPUTS:	**out** = nearest integer of numb towards zero

3.2 MY_IS NAN

This function determines whether the input number is NaN

INPUTS:	**numb** = number
OUTPUTS:	**out** = boolean (true if numb is NaN)

3.3 SIMPS1

This function applies the Simpson rule to evaluate integrals of the type

$$\int_a^b f(x)dx$$

INPUTS:	**fun** = integrand function in one variable

	a = lower bound of the integration interval **b** = upper bound of the integration interval **mx** = natural number: 2mx-1 is the number of subintervals considered in [a,b] to perform the integration
OUTPUTS:	**out** = value of the integral

3.4 SIMPS2

This function applies the Simpson rule to evaluate integrals of the type

$$\int_c^d \int_a^b f(x)dxdy$$

INPUTS:	**fun** = integrand function in two variables **a** = lower bound of the first integration interval **b** = upper bound of the first integration interval **c** = lower bound of the second integration interval **d** = upper bound of the second integration interval **mx** = natural number: 2mx-1 is the number of subintervals considered in [a,b] to perform the integration **my** = natural number: 2my-1 is the number of subintervals considered in [c,d] to perform the integration
OUTPUTS:	**out** = value of the integral

3.5 TRAPEZOIDALRULE

This function applies the trapezoidal rule to evaluate integrals of the type

$$\int_a^b f(x)dx$$

INPUTS:	**fun** = integrand function in one variable **a** = lower bound of the integration interval **b** = upper bound of the integration interval **nn** = number of subintervals considered in [a,b] to perform the integration
OUTPUTS:	**out** = value of the integral

3.6 TRAPEZOIDALRULEV2

This function applies the trapezoidal rule to evaluate integrals of the type

$$\int_a^b f(x)dx$$

INPUTS:	**xVect** = vector of points in the integration interval [a,b] **yVect** = vector of values of the integrand evaluated at xVect
OUTPUTS:	**out** = value of the integral

3.7 GETINTEGRAL

This function is an interface used to solve numerically integrals of the type

$$\int_a^b f(x)dx$$

Currently it depends on trapezoidalrule

INPUTS:	**fun** = integrand function in one variable **a** = lower bound of the integration interval **b** = upper bound of the integration interval **nn** = number of subintervals considered in [a,b] to perform the integration
OUTPUTS:	**out** = value of the integral

external packages.

4. MyElliptic.jl

This file contains all the function used to compute the elliptic integrals.

- DEPENDENCIES ON EXTERNAL PACKAGES: Elliptic, LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: functionslibrary.jl

4.1 ellP

This function computes the incomplete integral of third kind,

$$\int_0^{\phi} \frac{1}{(1+k\sin^2\theta)\sqrt{1-m\sin^2\theta}}d\theta$$

INPUTS:	**k** = characteristic **phi** = amplitude **m** = parameter
OUTPUTS:	**out** = value of the integral

This function depends on the the function ellPC, used compute the value of the integral when the parameter is negative.

4.2 ellF

This function computes the incomplete integral of first kind,

$$\int_0^{\phi} \frac{1}{\sqrt{1-m\sin^2\theta}}d\theta$$

INPUTS:	**phi** = amplitude **m** = parameter
OUTPUTS:	**out** = value of the integral

This function depends on the the function ellFC, used compute the value of the integral when the parameter is negative.

4.3 ellE

This function computes the incomplete integral of second kind,

$$\int_0^{\phi} \sqrt{1-m\sin^2\theta}\, d\theta$$

INPUTS:	**phi** = amplitude **m** = parameter
OUTPUTS:	**out** = value of the integral

This function depends on the the function ellEC, used compute the value of the integral when the parameter is negative.

5. POLYNOMIALEQUATIONS.JL

This file contains functions used to solve equations of second, third and fourth degrees.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: -

5.1 EQ2DEG

This function computes the real roots of an equation of second degree of the form $a_0x^2 + a_1x + a_2 = 0$

INPUTS:	**coeffs =** [a_0,a_1,a_2]
OUTPUTS:	**nreal** = number of real roots **xsol** = vector of length nreal with the real roots

5.2 EQ3DEG

This function computes the real roots of an equation of third degree of the form $a_0x^3 + a_1x^2 + a_2x + a_3 = 0$

INPUTS:	**coeffs =** [a_0,a_1,a_2,a_3]
OUTPUTS:	**nreal** = number of real roots **xsol** = vector of length nreal with the real roots

5.3 EQ4DEG

This function computes the real roots of an equation of third degree of the form $a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4 = 0$

INPUTS:	**coeffs =** [a_0,a_1,a_2,a_3,a_4]
OUTPUTS:	**nreal** = number of real roots **xsol** = vector of length nreal with the real roots

6. ASTROCONSTANTS.JL

This file contains a function “astroconstants” returning several astronomical constants.

- DEPENDENCIES ON EXTERNAL PACKAGES: -
- DEPENDENCIES ON INTERNAL FILES: -

6.1 ASTROCONSTANTS

This function returns a vector of constants relative to the celestial body associated to the identifier given as an input

INPUTS:	**in** = integer number used to identify the sets of constants
OUTPUTS:	in = 0 : Generic astronomical constants: **out** = [1: Universal gravity constant (G) (from DITAN and Horizon) [km^3/(kg*s^2)]; 2: Astronomical Unit (AU) (from DE405) [km]; 3: Solar radiation pressure at 1 au [kg/(s^2*m)] in = 1 : Mercury **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 2 : Venus **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 3 : Earth **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]; 3: Earth-Sun mean distance; 4: Earth magnetic dipole strenght [kg*km^3/(s^2*A)]; 5: Earth rotation mean motion [rad/s]; 6-9: zonal harmonics J2-J5] in = 4 : Mars **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 5 : Jupiter **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 6 : Saturn **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 7 : Uranus **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 8 : Neptune **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 9 : Pluto **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]]

	in = 10 : Sun **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]] in = 11 : Moon **out** = [1: Planetary constants of the planets (mu = mass * G) [km^3/s^2]; 2: Mean radius of the planets [km]; 3: Earth-Moon mean distance]

7. ATMOSPHERE.JL

This file contains functions returning two kinds of atmospheric models: the exponential atmospheric model and the nrlmsise00 atmospheric model.

- DEPENDENCIES ON EXTERNAL PACKAGES: SatelliteToolboxTransformations ; SatelliteToolboxAtmosphericModels; LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: timevarchange.jl;

7.1 EXPONENTIAL_ATM_MODEL

This function reads the exponential atmospheric model given in matrix form as an input and returns the atmospheric density at the required altitude.

INPUTS:	**h** = current altitude **A** = matrix containing atmospheric model in each row: [min altitude [km], density [kg/m^3], scale height [km]
OUTPUTS:	**rho** = density [kg/m^3]

7.2 GETEXPONENTIALATMOSPHERICDENSITYMODELEARTH

This function returns the exponential density model for the Earth in (Vallado, 1997) (Table 7-4)

INPUTS:	
OUTPUTS:	**A** = matrix containing atmospheric model in each row: [min altitude [km], density [kg/m^3], scale height [km]

7.3 GETATMOSPHEAREPROPERTIES_NRLMSISE00

This function return the atmospheric characteristics according to the nrlmsise00 model

INPUTS:	**mjd2000** = epoch in modified julian day 2000 **rV** = position vector with respect to the equatorial earth centred reference frame with the x-axis towards the vernal equinox **solarfluxdata** = table with solar flux data with format: [jd-AP1-AP2-AP3-AP4-AP5-AP6-AP7-AP8-AP9-AP_AVG-f107_OBS-f107_CENTER81-f107_CENTER90] with

	AP1 : Planetary Equivalent Amplitude (Ap) for 0000-0300 UT. AP2 : Planetary Equivalent Amplitude (Ap) for 0300-0600 UT. AP3 : Planetary Equivalent Amplitude (Ap) for 0600-0900 UT. AP4 : Planetary Equivalent Amplitude (Ap) for 0900-1200 UT. AP5 : Planetary Equivalent Amplitude (Ap) for 1200-1500 UT. AP6 : Planetary Equivalent Amplitude (Ap) for 1500-1800 UT. AP7 : Planetary Equivalent Amplitude (Ap) for 1800-2100 UT. AP8 : Planetary Equivalent Amplitude (Ap) for 2100-0000 UT. AP_AVG : Arithmetic average of the 8 Ap indices for the day. F10.7_OBS : Observed 10.7-cm Solar Radio Flux (F10.7). Measured at Ottawa at 1700 UT daily from 1947 Feb 14 until 1991 May 31 and measured at Penticton at 2000 UT from 1991 Jun 01 on. Expressed in units of 10-22 W/m2/Hz. F10.7_CENTER81 : Centered 81-day arithmetic average of F10.7. F10.7_CENTER90 : Centered 90-day arithmetic average of F10.7.
OUTPUTS:	**rhoinf** = density [kg/m^3] **Tinf** = temperature [K] **ainf** = sound velocity [m/s] **ndens** = number densities of He, O, N2, O2, Ar, H, N **Te** = exospheric temperature

This function depends on a further function contained in the same file, i.e. getSolarFluxAndGeomagneticAp(jd,solarflux) which reads the matrix solarfluxdata and returns the data required to compute the atmospheric model at the required julian day jd.

8. ATTITUDEVARCHANGE.JL

This file contains all the functions required to perform the conversion of attitude variables.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: rotationmatrixes.jl

8.1 EULER2EULERCANONICAL.JL

This function performs the transformation (euler angles 313, angular velocity) → (Euler canonical variables)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let omV = [p,q,r]^T be the angular velocity in the body reference frame.
Let N' be the node between the XY and xy planes.

INPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame.

	IV = [A,B,C]
OUTPUTS:	**eulervar** = [pphi, ptheta, ppsi,phi, theta, psi] where pphi = momentum conjugated to phi (=A*p*sin(psi)*sin(theta)+B*q*cos(psi)*sin(theta)+C*r*cos(theta)) ptheta = momentum conjugated to theta (=A*p*cos(psi)-B*q*sin(psi) ppsi = momentum conjugated to psi (=C*r)

8.2 EULERCANONICAL2EULER

This function performs the transformation (euler canonical variables)→ (euler angles 313, angular velocity)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let omV = [p,q,r]^T be the angular velocity in the body reference frame.
Let N' be the node between the XY and xy planes.

INPUTS:	**eulervar** = [pphi, ptheta, ppsi,phi, theta, psi] where pphi = momentum conjugated to phi (=A*p*sin(psi)*sin(theta)+B*q*cos(psi)*sin(theta)+C*r*cos(theta)) ptheta = momentum conjugated to theta (=A*p*cos(psi)-B*q*sin(psi) ppsi = momentum conjugated to psi (=C*r)where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame. **IV** = [A,B,C]
OUTPUTS:	**euanglesV =** [phi,theta,psi] **omV** = [p,q,r], angular velocity in the body reference frame.

8.3 EULER2QUAT

This function performs the transformation (euler angles 313) →(quaternions)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let N' be the node between the XY and xy planes.

INPUTS:	**euangles =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis
OUTPUTS:	**quat** = [q1,q2,q3,q4], quaternions

8.4 QUAT2EULER

This function performs the transformation (quaternions) →(euler angles 313)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let N' be the node between the XY and xy planes.

INPUTS:	**quat** = [q1,q2,q3,q4], quaternions
OUTPUTS:	**euangles =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis

8.5 EULER2TAITBRYAN

This function performs the transformation (euler angles 313) →(tait-bryan angles)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let N' be the node between the XY and xy planes.

INPUTS:	**euangles =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis
OUTPUTS:	**tb** = [yaw,pitch,roll], tait-bryan angles

8.6 TAITBRYAN2EULER

This function performs the transformation (tait-bryan angles) →(euler angles 313)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let N' be the node between the XY and xy planes.

INPUTS:	**tb** = [yaw,pitch,roll], tait-bryan angles
OUTPUTS:	**euangles =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis

8.7 EULERCANONICAL2ANDOYER

This function performs the transformation (euler canonical variables)→ (andoyer-serret variables)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV = [p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**eulervar** = [pphi, ptheta, ppsi,phi, theta, psi] where pphi = momentum conjugated to phi (=A*p*sin(psi)*sin(theta)+B*q*cos(psi)*sin(theta)+C*r*cos(theta)) ptheta = momentum conjugated to theta (=A*p*cos(psi)-B*q*sin(psi) ppsi = momentum conjugated to psi (=C*r) phi = angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis
OUTPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N

8.8 ADOYER2EULERCANONICAL

This function performs the transformation (andoyer-serret variables)→ (euler canonical variables)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV = [p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**andoyervar** = [L,G,H,l,g,h] where

	L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N
OUTPUTS:	**eulervar** = [pphi, ptheta, ppsi,phi, theta, psi] where pphi = momentum conjugated to phi (=A*p*sin(psi)*sin(theta)+B*q*cos(psi)*sin(theta)+C*r*cos(theta)) ptheta = momentum conjugated to theta (=A*p*cos(psi)-B*q*sin(psi) ppsi = momentum conjugated to psi (=C*r) phi = angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis

8.9 EULER2ANDOYER

This function performs the transformation (euler angles 313, angular velocity)→ (andoyer-serret variables)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV = [p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame. **IV** = [A,B,C]
OUTPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N

	IV = [A,B,C]

8.10 ANDOYER2EULER

This function performs the transformation (andoyer-serret variables)→ (euler angles 313, angular velocity)

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV =[p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N **IV** = [A,B,C]
OUTPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame.

8.11 EULER2SADOV

This function performs the transformation (euler313, angular velocity) → (sadov variables)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy

Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame. **IV** = [A,B,C]
OUTPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k).

This function depends on a further function included in the same file, m2JlJgratioV2, used to compute the Sadov variable Jl.

8.12 SADOV2EULERANDJL

This function performs the transformation (sadov variables)→(euler313 + angular velocity)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).

Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadovvar** = [Jg,Jh,psil,psig,psih] with Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k) **IV** = [A,B,C]
OUTPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame. **Jl** = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m))

This function depends on a further function included in the same file, m2JlJgratioV2, used to compute the Sadov variable Jl.

8.13 SADOV2EULER

This function performs the transformation (sadov variables)→(euler313 + angular velocity)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with

	Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k) **IV** = [A,B,C]
OUTPUTS:	**euanglesV =** [phi,theta,psi] where phi **=** angle between X axis and N' theta = inclination between the XY plane and the xy plane psi = angle N' and x axis **omV** = [p,q,r], angular velocity in the body reference frame.

This function depends on sadov2eulerandJl.

8.14 ANDOYER2SADOV

This function performs the transformation (andoyer-serret variables)→(serret variables)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV.

	l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N **IV** = [A,B,C]
OUTPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k).

8.15 SADOV2ANDOYERV1

This function performs the transformation (sadov variables)→(andoyer-serret variables)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis

	psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k). **IV** = [A,B,C]
OUTPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N

8.16 SADOV2ANDOYERV2

This function performs the transformation (sadov variables)→(andoyer-serret variables)

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4)

	IV = [A,B,C]
OUTPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N

This function depends on a function included in the file, JlJgratio2m_bisection, used to compute the value of skm=k/(m+k), required to determine the Andoyer-Serret variables. This last function JlJgratio2m_bisection depends on a further function included in the file m2JlJgratioV1. Finally, sadov2andoyerV2 also depends on sadov2andoyerV1.

As an alternative to JlJgratio2m_bisection one could use JlJgratio2m_newton (also in this file). Currently, JlJgratio2m_newton is not used.

8.17 SADOV2SADOVLIKE

This function performs the transformation (sadov variables)→(sadov-like variables)
** The Sadov-like variables are used to erase a singularity in the attitude equations of motion expressed in Sadov variables, occurring when the angular momentum is aligned with the inertial Z axis

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m)) Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV.

	psil = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4) **skm** = k/(m+k).
OUTPUTS:	**sadov-like variables** = [J1,J2,J3,J4,J5,J6,J7] with J1 = skm J2 = Jg J3 = Jh J4 = psil J5 = psig + Jh/Jg*psih J6 = sqrt(Jg^2-Jh^2)*cos(psih) J7 = sqrt(Jg^2-Jh^2)*sin(psih)

8.18 SADOVLIKE2SADOV

This function performs the transformation (sadov-like variables)→(sadov variables)
** The Sadov-like variables are used to erase a singularity in the attitude equations of motion expressed in Sadov variables, occurring when the angular momentum is aligned with the inertial Z axis

Consider an inertial reference frame XYZ.
Consider a body reference frame xyz in principal axes of inertia such that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let omV = [p,q,r]^T be the angular velocity of the satellite in the body frame
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**sadov-like variables** = [J1,J2,J3,J4,J5,J6,J7] with J1 = skm J2 = Jg J3 = Jh J4 = psil J5 = psig + Jh/Jg*psih J6 = sqrt(Jg^2-Jh^2)*cos(psih) J7 = sqrt(Jg^2-Jh^2)*sin(psih)

	Where skm **=** k/(m+k). Jg = G Jh = Gcos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. psiI = pi/2 ellF(pi/2,m)ellF(lambda,m), lambda = atan(cos(I),sqrt{1+k}sin(I)), I = angle between N'' and x axis psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), g = angle between N and N'' psih = angle between X axis and N ** ellF, ellP elliptic integrals (see Section 4)
OUTPUTS:	**sadovvar** = [Jl,Jg,Jh,psil,psig,psih] with Jl = 2G/pi sqrt{k+m} sqrt{(1+k)/k} (ellP(-k,pi/2,m)-m/(k+m)ellF(pi/2,m))

8.19 ANDOYER2ANDOYERLIKE

This function performs the transformation (andoyer-serret variables)→(andoyer-like variables)
** The Andoyer-like variables are used to erase a singularity in the attitude equations of motion expressed in Andoyer variables, occurring when the angular momentum is aligned with the inertial Z axis

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV =[p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**andoyervar** = [L,G,H,l,g,h] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N
OUTPUTS:	**andoyer-like variables** = [J1,J2,J3,J4,J5,J6,J7] with J1 = L J2 = G J3 = H J4 = l J5 = g + H/G*h J6 = sqrt(G^2-H^2)*cos(h) J7 = sqrt(G^2-H^2)*sin(h)

8.20 ANDOYERLIKE2ANDOYER

This function performs the transformation (andoyer-serret variables)→(andoyer-like variables)
** The Andoyer-like variables are used to erase a singularity in the attitude equations of motion expressed in Andoyer variables, occurring when the angular momentum is aligned with the inertial Z axis

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV =[p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**andoyer-like variables** = [J1,J2,J3,J4,J5,J6,J7] with J1 = L J2 = G J3 = H J4 = l J5 = g + H/G*h J6 = sqrt(G^2-H^2)*cos(h) J7 = sqrt(G^2-H^2)*sin(h) where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N
OUTPUTS:	**andoyervar** = [L,G,H,l,g,h]

9. ORBITVARCHANGE.JL

This file contains all the functions required to perform the conversion of attitude variables.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: -

9.1 KEP2CAR

This function performs the transformation (keplerian elements)→(cartesian coordinates)

INPUTS:	**kepV** = [a,e,i,o,O,nu] with a = semi-major axis e = eccentricity

	i = inclination [rad] o = argumentum of pericentre [rad] O = right ascension of the ascending node [rad] nu = true anomaly [rad] **mu** = planetary constant !! the units of measurement of a and mu must be coherent
OUTPUTS:	**carV** = [v,x] with v = velocity vector x = position vector !! the units of x and v are coherent with those of a and mu

9.2 CAR2KEP

This function performs the transformation (cartesian coordinates) →(keplerian elements)

INPUTS:	**carV** = [v,x] with v = velocity vector x = position vector **mu** = planetary constant !! the units of measurement of x,v and mu must be coherent
OUTPUTS:	**kepV** = [a,e,i,o,O,nu] with a = semi-major axis e = eccentricity i = inclination [rad] o = argumentum of pericentre [rad] O = right ascension of the ascending node [rad] nu = true anomaly [rad] !! a has the same unit of measuremet as x

9.3 KEP2EQUI

This function performs the transformation (keplerian elements)→(equinoctial coordinates)

INPUTS:	**kepV** = [a,e,i,o,O,nu] with a = semi-major axis e = eccentricity i = inclination [rad] o = argumentum of pericentre [rad] O = right ascension of the ascending node [rad] nu = true anomaly [rad] **idxLL** = identifier of the type of longitude required in the outputs idxLL = 1 : true longitude (nu+o+O) idxLL = 2 : eccentric longitude (E+o+O), E=eccentric anomaly idxLL = 3 : mean longitude (M+o+O), M = mean anomaly
OUTPUTS:	**equiV** = [a,P1,P2,Q1,Q2,LL]

	with P1 = e*sin(o+O) P2 = e*cos(o+O) Q1 = tan(i/2)*sin(O) Q2 = tan(i/2)*cos(O) LL = nu+o+O if idxLL=1 E+o+O if idxLL =2 M+o+O if idxLL = 3

9.4 EQUI2KEP

This function performs the transformation (equinoctial elements)→(Keplerian elements)

INPUTS:	**equiV** = [a,P1,P2,Q1,Q2,LL] with P1 = e*sin(o+O) P2 = e*cos(o+O) Q1 = tan(i/2)*sin(O) Q2 = tan(i/2)*cos(O) LL = nu+o+O if idxLL=1 E+o+O if idxLL =2 M+o+O if idxLL = 3 where a = semi-major axis e = eccentricity i = inclination o = argumentum of pericentre O = right ascension of the ascending node nu = true anomaly E=eccentric anomaly M = mean anomaly **idxLL** = identifier of the type of longitude
OUTPUTS:	**kepV** = [a,e,i,o,O,nu] i,o,O,nu in [rad]

9.5 TRUELONG2ECCLONG

This function performs the conversion from true longitude to eccentric longitude

INPUTS:	**TL** = true longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**EL** = eccentric longitude

9.6 ECCLONG2TRUELONG

This function perform the conversion from eccentric longitude to true longitude

INPUTS:	**EL** = eccentric longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**TL** = true longitude

9.7 MEANLONG2ECCLONG

This function performs the conversion from mean longitude to eccentric longitude

INPUTS:	**ML** = mean longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**EL** = eccentric longitude

9.8 ECCLONG2MEANLONG

This function performs the conversion from eccentric longitude to mean longitude

INPUTS:	**EL** = eccentric longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**ML** = mean longitude

9.9 MEANLONG2TRUELONG

This function performs the conversion from mean longitude to true longitude

INPUTS:	**ML** = mean longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node

OUTPUTS:	**TL** = true longitude

9.10 TRUELONG2MEANLONG

This function performs the conversion from true longitude to mean longitude

INPUTS:	**TL** = true longitude **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**ML** = mean longitude

9.11 MEAN2TRUELONGITUDE

This function performs the conversion from mean anomaly to true longitude

INPUTS:	**P1** = e*sin(o+O) **P2** = e*cos(o+O) **Q1** = tan(i/2)*sin(O) **Q2** = tan(i/2)*cos(O) **M** = mean anomaly with e = eccentricity i = inclination o = argumentum of pericentre O = right ascension of the ascending node
OUTPUTS:	**TL** = true longitude

9.12 TRUE2MEAN

This function performs the conversion from true to mean anomaly, in case of elliptic orbits

INPUTS:	**nu** = true anomaly **e** = eccentricity
OUTPUTS:	**M** = mean anomaly

9.13 ECCENTRIC2MEAN

This function performs the conversion from eccentric to mean anomaly

INPUTS:	**nu** = true anomaly **e** = eccentricity
OUTPUTS:	**M** = mean anomaly

9.14 MEAN2TRUEANDECCENTRIC

This function perform the conversion from mean to true and eccentric anomalies, in the case of elliptic orbits (if the eccentricity is strictly smaller than 1).

INPUTS:	**M** = mean anomaly **e** = eccentricity
OUTPUTS:	**nu** = true anomaly **E** = eccentric anomaly **ier** = control variable: if the conversion succeded it is equal to 1

This function depends on the following functions contained in the file: Sfun, DSfun, ellipticnearparabolic, ellipticstrong. The conversion is based on the algorithm described by Farnocchia, Cioci, Milani (2012).

9.15 EQUI2EQUATIONOFTHECENTRE

This function returns the value of the equation of the centre

INPUTS:	**TL** = nu+o+O **Q1** = tan(i/2)*sin(O) **Q2** = tan(i/2)*cos(O) **P1** = e*sin(o+O) **P2** = e*cos(o+O) with e = eccentricity i = inclination o = argumentum of pericentre O = right ascension of the ascending node nu = true anomaly
OUTPUTS:	**out** = equation of the centre, (=nu-M, M=mean anomaly)

10. ROTATIONMATRICES.JL

This function contains all the functions to compute the rotation matrices defining the attitude and the rotation matrices from inertial to perifocal/radial-transversal-normal reference frames. It contains also functions which return some attitude variables (euler313, tait-bryan, quaternions) given the rotation matrix defining the attitude.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearAlgebra
- DEPENDENCIES ON INTERNAL FILES: -

10.1 EULER2IBROTMAT

This function returns the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

INPUTS:	**euanglesV =** [phi,theta,psi], Euler angle 313
OUTPUTS:	**R** = rotation matrix

10.2 SADOV2IBROTMAT

This function returns the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

Consider an inertial reference Frame XYZ.
Consider a body-fixed reference frame xyz in principal axes of inertia.

The body reference frame is such that m belongs to [0,1] and k>0 where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C the moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the rotational kinetic energy
Let N' be the node between the XY and xy planes.

INPUTS:	**k** **k1r** = sqrt(1+k) **skmR** = sqrt(k/(m+k)) **smkR** = sqrt(m/(m+k)) **cn** = JacobiCos(2/pi*psil*ellF(pi/2.0,m),m) **sn** = JacobiSin(2/pi*psil*ellF(pi/2.0,m),m) **dn** = sqrt(1-m*sn^2.0) **cg** = cos(g), g = psig + sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2.0,m)psil*2.0/pi-ellP(-k,JacobiAM(2/pi*psil*ellF(pi/2.0,m),m),m)) **sg** = sin(g) **cpsih** = cos(psih) **spsih** = sin(psih) **cdelta** = Jh/Jg **sdelta** = sqrt(1-Jh^2.0/Jg^2.0) with [Jl,Jg,Jh,psil,psig,psih] = sadov variables
OUTPUTS:	**R** = rotation matrix **Rpsilpsig** = rotation matrix from reference frame having a Z axis aligned with the angular momentum to the rotating frame

10.3 ANDOYER2IBROTMAT

This function returns the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

INPUTS:	**andoyervar** = [L,G,H,l,g,h], Andoyer-Serret variables
OUTPUTS:	**R** = rotation matrix

10.4 TAITBRYAN2IBROTMAT

This function returns the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

INPUTS:	**tb** = [yaw,pitch,roll], tait-bryan angles
OUTPUTS:	**R** = rotation matrix

10.5 QUAT2BIROTMAT

This function returns the rotation matrix from the rotating frame co-moving with the rigid body to the inertial frame

INPUTS:	**quat** = [q1,q2,q3,q4], quaternions
OUTPUTS:	**R** = rotation matrix

10.6 ROTMATIB2EULER

This function computes the euler angles 313 from the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

INPUTS:	**R** = rotation matrix
OUTPUTS:	**euanglesV =** [phi,theta,psi], Euler angle 313

10.7 ROTMATIB2TAITBRYAN

This function computes the tait-bryan angles from the rotation matrix from inertial frame to rotating frame co-moving with the rigid body.

INPUTS:	**R** = rotation matrix
OUTPUTS:	**tb** = [yaw,pitch,roll], tait-bryan angles

10.8 ROTMATBI2QUAT

This function computes the quaternions from the rotation matrix from rotating frame co-moving with the rigid body to inertial frame

INPUTS:	**R** = rotation matrix
OUTPUTS:	**quat** = [q1,q2,q3,q4], quaternions

10.9 KEP2IPROTMAT

This function returns the rotation matrix from inertial to perifocal reference frame

INPUTS:	**i** = inclination [rad] **O** = right ascension of the ascending node [rad] **o** = argument of pericentre [rad]
OUTPUTS:	**R** = rotation matrix

10.10 EQUI2ROTMATIRTH

This function returns the rotation matrix from inertial radial-transversal-normal reference frame

INPUTS:	**Q1** = tan(i/2)*sin(O) **Q2** = tan(i/2)*cos(O) **sTL** = sin(TL) **cTL** = cos(TL) with i = inclination [rad] O = right ascension of the ascending node [rad] TL = true longitude
OUTPUTS:	**R** = rotation matrix

11. TIMEVARCHANGE.JL

This file contains all the functions required to perform the conversion of epoch measurement systems

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra
- DEPENDENCIES ON INTERNAL FILES: -

NB: translation in Julia from Calypso

11.1 HMS2FRACDAY

This function converts hours, minutes and seconds to the fraction of day

INPUTS:	**hrs** = number of hours an integer greater or equal to 0 and lower or equal to 23 **mn** = number of minutes as integer greater or equal to 0 and lower or equal to 59. **sec** = number of seconds as a real greater or equal to 0 and strictly lower than 60.
OUTPUTS:	**fracDay** = A single real greater or equal to 0 and strictly lower than 1

11.2 FRACDAY2HMS

This function converts the fraction of day to hours, minutes and seconds

INPUTS:	**fracDay** = A single real greater or equal to 0 and strictly lower than
OUTPUTS:	1**hrs** = number of hours an integer greater or equal to 0 and lower or equal to 23 **mn** = number of minutes as integer greater or equal to 0 and lower or equal to 59. **sec** = number of seconds as a real greater or equal to 0 and strictly lower than 60.

11.3 DATE2JD

This function returns the Julian day number of the given date (Gregorian calendar) plus a fractional part depending on the time of day.

Note: The function is valid for the whole range of dates since 12:00 noon 24 November -4713, Gregorian calendar. (This bound is set in order to have symmetry with the inverse function jd2date)
Note: The inputs must be feasible (i.e. the date must exist!). If an unfeasible date is inputed, wrong results are given because no check is done on that.

INPUTS:	**date** = Date in the Gregorian calendar, as a 6-elements vector [year, month, day, hour, minute, second].

OUTPUTS:	**jd** = Date in Julian Day. The JD (Julian day) count is from 0 at 12:00 noon, 1 January -4712 (4713 BC), Julian proleptic calendar. The corresponding date in Gregorian calendar is 12:00 noon, 24 November -4713.

11.4 JD2DATE

This function returns the Gregorian calendar date corresponding to a Julian day

INPUTS:	**jd** = Date in Julian Day. The JD (Julian day) count is from 0 at 12:00 noon, 1 January -4712 (4713 BC), Julian proleptic calendar. The corresponding date in Gregorian calendar is 12:00 noon, 24 November -4713. It must be non-negative
OUTPUTS:	**date** = Date in the Gregorian calendar, as a 6-elements vector [year, month, day, hour, minute, second].

11.5 DATE2MJD2000

This function returns the modified Julian day 2000 number of the given date (Gregorian calendar) plus a fractional part depending on the time of day.

Note: The function is valid for the whole range of dates since 12:00 noon 24 November -4713, Gregorian calendar. (This bound is set in order to have symmetry with the inverse function jd2date)
Note: The inputs must be feasible (i.e. the date must exist!). If an unfeasible date is inputed, wrong results are given because no check is done on that.

INPUTS:	**date** = Date in the Gregorian calendar, as a 6-elements vector [year, month, day, hour, minute, second].
OUTPUTS:	**mjd2000** = Date in modified Julian Day 2000. The modified julian day 2000 is defined as the number of days since 01-01-2000, 12:00 noon

11.6 MJD20002DATE

This function returns the Gregorian calendar date corresponding to a Modified Julian day 2000

INPUTS:	**mjd2000** = Date in modified Julian Day 2000. The modified julian day 2000 is defined as the number of days since 01-01-2000, 12:00 noon
OUTPUTS:	**date** = Date in the Gregorian calendar, as a 6-elements vector [year, month, day, hour, minute, second].

11.7 MJD20002JD

This function returns the Julian day number corresponding to the given modified Julian day 2000 number

INPUTS:	**mjd2000** = Date in modified Julian Day 2000. The modified julian day 2000 is defined as the number of days since 01-01-2000, 12:00 noon
OUTPUTS:	**jd** = Date in Julian Day. The JD (Julian day) count is from 0 at 12:00 noon, 1 January -4712 (4713 BC), Julian proleptic calendar. The corresponding date in Gregorian calendar is 12:00 noon, 24 November -4713.

11.8 JD2MJD2000

This function returns the modified Julian day 2000 number corresponding to the given Julian day number

INPUTS:	**jd** = Date in Julian Day. The JD (Julian day) count is from 0 at 12:00 noon, 1 January -4712 (4713 BC), Julian proleptic calendar. The corresponding date in Gregorian calendar is 12:00 noon, 24 November -4713.
OUTPUTS:	**mjd2000** = Date in modified Julian Day 2000. The modified julian day 2000 is defined as the number of days since 01-01-2000, 12:00 noon

12. EPHEMERIS.JL

This file contains the functions to compute the ephemeris of several bodies in the Solar System.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra
- DEPENDENCIES ON INTERNAL FILES: orbitvarchange.jl

NB: translation in Julia from Calypso.

12.1 CELESTIALBODIESEPHEMERIS_KEP

This function computes the analytical ephemerides for celestial bodies.
Planetay orbital elements are restituited in a Sun-centred ecliptic system. These ephemerides were succesfully compared with JPL/NAIF/SPICE ephemerides using de405.bps. Lunar orbital elements are restituited in an Earth-centered equatorial reference frame.

INPUTS:	mjd2000 = epoch in modified Julian day 2000 ibody = integer number identifying the celestial body (< 11) ibody = 1: Mercury ibody = 2: Venus ibody = 3: Earth ibody = 4: Mars ibody = 5: Jupiter ibody = 6: Saturn ibody = 7: Uranus ibody = 8: Neptune ibody = 9: Pluto ibody = 10: Sun ibody = 11: Moon

OUTPUTS:	kep = [a e i Om om nu], [km, rad], mean Keplerian elements of date

This function depends on the function eph_moon in this file.

12.2 CELESTIALBODIESEPHEMERIS_POSITION

This function returns the mean position of the celestial bodies. The position vectors are restitued in a Sun-centred ecliptic system for planets, and in an Earth-centred equatorial reference frame for the Moon.

INPUTS:	mjd2000 = epoch in modified Julian day 2000 ibody = integer number identifying the celestial body (< 11) ibody = 1: Mercury ibody = 2: Venus ibody = 3: Earth ibody = 4: Mars ibody = 5: Jupiter ibody = 6: Saturn ibody = 7: Uranus ibody = 8: Neptune ibody = 9: Pluto ibody = 10: Sun ibody = 11: Moon
OUTPUTS:	**rV** =position vector

This function depends on the functions lunar_position_equatorialrefframe and eph_moon contained in this file, beside celestialbodiesephemeris_kep.

13. ECLIPSE.JL

This file contains the functions used to compute the entrance and the exit of a space body in the region of shadow of the central planet it is orbiting. The shadow is modelled as a cylindrical region.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra
- DEPENDENCIES ON INTERNAL FILES: polynomalequations.jl;

13.1 ECLIPSEINOUT

This function returns the true longitude at the entrance and exit from the shadow region assuming a cylindrical model for this last one

INPUTS:	**rsun** = position vector of the Sun wrt to the central planet [km] **sma** = sei-major axis [km] **P1** = e*sin(O+o) **P2** = e*cos(O+o) **q11** = (1-Q1^2+Q2^2)/GG, **q12** = (2*Q1*Q2)/GG; **q13** = (-2*Q1)/GG; **q21** = (2*Q1*Q2)/GG; **q22** = (1+Q1^2-Q2^2)/GG; **q23** = (2*Q2)/GG; with e = eccentricity

	O = longitude of ascending node o = argument of pericentre GG = 1+Q1^2+Q2^2, Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), i = inclination **rPlanet** = mean radius of the central planet
OUTPUTS:	**inshadow** = boolean, true if the orbit is shadowed; false if the orbit is always in sunlight **TLin** = true longitude at entrance of the shadow region (if inshadow=false, TLin=NaN) [rad] **TLout** = true longitude at the exit from the shadow region (if inshadow=false, TLout=NaN) [rad] true longitude = O + o + nu, nu = true anomaly

13.2 ECLIPSEINOUTV2

This function returns the true longitude at the entrance and exit from the shadow region assuming a cylindrical model for this last one

INPUTS:	**rsun** = position vector of the Sun wrt to the central planet [km] **sma** = sei-major axis [km] **P1** = e*sin(O+o) **P2** = e*cos(O+o) **Q1** = tan(i/2)*sin(OM), **Q2** = tan(i/2)*cos(OM), with e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **rPlanet** = mean radius of the central planet
OUTPUTS:	**inshadow** = boolean, true if the orbit is shadowed; false if the orbit is always in sunlight **TLin** = true longitude at entrance of the shadow region (if inshadow=false, TLin=NaN) [rad] **TLout** = true longitude at the exit from the shadow region (if inshadow=false, TLout=NaN) [rad] true longitude = O + o + nu, nu = true anomaly

14. EQMOTION.JL

This file contains the function returning the system of ordinary differential equations describing the full attitude-orbital dynamics of a rigid body orbiting a planet.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra; Special Functions; HypergeometricFunctions
- DEPENDENCIES ON INTERNAL FILES: atmosphere.jl; ephemeris.jl;

The environmental perturbations considered are:

- Gravity-gradient torque
- Residual magnetic torque
- Light pressure torque
- Atmospheric drag torque
- Zonal harmonics J2, J3, J4, J5
- Third body acceleration (ex. Lunar gravity for a body orbiting the Earth)
- Sun gravity
- Light pressure acceleration
- Atmospheric drag acceleration

Concerning the atmospheric drag perturbation, two different models are implemented, one associated to the exponential model of the atmosphere and one associated to the nrlmsise00 model.

For further details see next Section Propagators_girf.jl

14.1 ORBITATTITUDEEV_QUAT

This function returns the ODE describing the coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are the components of the angular velocity and the quaternions

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [a,P1,P2,Q1,Q2,ML,q1,q2,q3,q4,p,q,r] where a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), [q1,q2,q3,q4] = quaternions [p,q,r] = angular velocity with e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **p** = array with parameters and settings (*** see Section 15.1) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the functions getgravitytorque, getmagnetictorque, getsrptorqueandforce, getdragtorqueandforce_atmexp/getdragtorqueandforce_atmnrlmsise00, getCentralBodyPotentialAcc, get3BodyAcc, getSunGravityPert returning the perturbing torques and forces/accelerations.

14.2 ORBITATTITUDEEV_SADOV

This function returns the ODE describing the coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are (skm,Jg,Jh,psil,psig,psih).

The equations of motion have singularity. This function is DEPRECATED.

Consider an inertial reference framx XYZ and a rotating reference frame xyz co-moving with the satellite.
The body reference frame xyz selected must be such that that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [a,P1,P2,Q1,Q2,ML,skm,Jg,Jh,psil,psig,psih] where a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), ML = mean longitude skm = k/(m+k) Jg = G Jh = G*cos(delta), psil = pi/2 ellF(pi/2,m)ellF(lambda,m), psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), psih = angle between X axis and N with e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination delta = inclination angle between the XY plane and the plane perpendicular to GV. lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis g = angle between N and N''

	** ellF, ellP elliptic integrals (see Section 4) **p** = = array with parameters and settings (*** see Section 15.1) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the functions getgravitytorque, getmagnetictorque, getsrptorqueandforce, getdragtorqueandforce_atmexp/getdragtorqueandforce_atmnrlmsise00, getCentralBodyPotentialAcc, get3BodyAcc, getSunGravityPert returning the perturbing torques and forces/accelerations.

15. PROPAGATORS_GIRF.JL

This file contains the propagators of the full coupled orbital and attitude dynamics. More specifically, it contains a propagator which employs quaternions and angular velocity as attitude variables and a propagator which uses Sadov variables. As the equations of motion expressed in Sadov variables have singularities, the use of this second propagator is deprecated.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: orbitvarchange.jl; attitudevarchange.j; eqMotion.jl; astroConstants.jl

Two reference frame are considered. An generic inertial reference frame girf and a rotating reference frame Oxyz co-moving with the satellite. The frame Oxyz **must** be centred in the satellite centre of mass and must be in principal axes, such that the principal moments of inertia are A<=B<=C, with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.

The environmental perturbations considered are:

- Gravity-gradient torque
- Residual magnetic torque
- Light pressure torque
- Atmospheric drag torque
- Zonal harmonics J2, J3, J4, J5
- Third body acceleration (ex. Lunar gravity for a body orbiting the Earth)
- Sun gravity
- Light pressure acceleration
- Atmospheric drag acceleration

The gravity-gradient torque is modelled as

$$\boldsymbol{M}_g = \frac{3\mu_P}{r^3}\begin{bmatrix}(C-B)\alpha_2\alpha_3\\(A-C)\alpha_3\alpha_1\\(B-A)\alpha_1\alpha_2\end{bmatrix}$$

where r is the planetocentric distance of the satellite's centre of mass, μ_P is the planet's gravitational parameter and $(\alpha_1, \alpha_2, \alpha_3)$ are the direction cosines of the position vector of the satellite's centre of mass expressed in Oxyz.

The magnetic torque is modelled as

$$M_m = I \times R_{i2b}B$$

Where I is the intrinsic magnetic moment of the satellite (expressed in Oxyz), R_{i2b} is the rotation matrix from girf to Oxyz and

$$B = \frac{\mu_m}{r^3}\left(\boldsymbol{e}_z - \frac{3\left(\boldsymbol{e}_Z \cdot \frac{\boldsymbol{r}}{r}\right)\boldsymbol{r}}{r}\right)$$

where μ_m is the magnetic dipole strength of the planet, $\boldsymbol{e}_z$ is the unit vector in the direction of the magnetic field dipole (expressed in girf) and $\boldsymbol{r}$ is the position vector of the satellite centre of mass (expressed in girf).

NB: the propagator is thought for satellites orbiting the Earth, so $\boldsymbol{e}_z$ is approximated as the unit vector in the direction of the Earth's north pole. The code may need to be corrected if this assumption does not hold for a different central body.

Both the light pressure and the atmospheric drag perturbations are based on a subdivision of the external surface of the rigid body in either triangular or squared facets. The total perturbation is given by the sum of the perturbations acting on each facets.

The light pressure force is

$$\boldsymbol{f}_{srp} = -P_{srp}\left(\frac{1\ au}{|\boldsymbol{r}_\odot - \boldsymbol{r}|}\right)^2 \sum_{j=1}^{n_f} S_j\left(c_{a,j}\boldsymbol{u} + c_{d,j}\boldsymbol{n}_j + c_{s,j}(\boldsymbol{u}\cdot\boldsymbol{n}_j)\boldsymbol{n}_j\right)\max(\boldsymbol{u}\cdot\boldsymbol{n}_j, 0)$$

with

P_{srp}	Solar radiation pressure at 1 au
$\boldsymbol{r}_\odot$	Sun planetocentric position vector expressed in girf
$\boldsymbol{u}$	$\frac{\boldsymbol{r}_\odot - \boldsymbol{r}}{\lvert\boldsymbol{r}_\odot - \boldsymbol{r}\rvert}$
$\boldsymbol{n}_j$	Outward normal unit vector of the j-th facet
S_j	Area of the j-th facet
n_f	Number of facets
$c_{a,j}$	$1 - \rho_j s_j$, ρ_j total reflectivity of the j-th facet and s_j fraction of ρ_j which is specular
$c_{d,j}$	$\frac{2}{3}(1 - s_j)\rho_j$
$c_{s,j}$	$2\rho_j s_j$

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the light pressure force acting of the j-th facet, where $\boldsymbol{\rho}_j$ is the vector from the centre of mass to the centroid of the facet.

Concerning the atmospheric drag perturbation, two different models are implemented. The first one is based on the exponential model of the atmosphere. It assumes that the drag force is in the direction of the relative incident stream and that the non-dimensional drag coefficient CD is constant, which has to be set by the user as a characteristic of the satellite. The drag force is modelled as

$$\boldsymbol{f}_d = -\frac{1}{2}\rho_\infty v_\infty^2 C_D \sum_{j=1}^{n_f} S_j \left(\frac{1}{3\pi} + \frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{2} + \frac{4}{3\pi}\left(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)^2\right)\boldsymbol{e}_\infty$$

where

ρ_∞	Free stream density
v_∞	Modulus of the aerodynamic flow velocity vector
$\boldsymbol{e}_\infty$	Outward direction of the aerodynamic flow velocity vector

The second model is based on the nrlmsise00 model of the atmosphere. The drag force has also a lift component and the non-dimensional drag and lift coefficients depend on the attitude and the Mach number of the rigid body. The drag force is modelled as

$$\boldsymbol{f}_{aero} = \frac{1}{2}\rho_\infty v_\infty^2 \sum_{j=1}^{n_f} S_j (C_{n,j}\boldsymbol{n}_j + C_{v,j}\boldsymbol{e}_\infty)$$

with

$$C_{V_j} = \left(\frac{\exp\left(-S_\infty^2\left(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)^2\right)}{S_\infty\sqrt{\pi}} + \left(1 + \operatorname{erf}\left(S_\infty\left(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)\right)\right)\left(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)\right)\max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{\left|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right|}, 0\right)$$

$$C_{N_j} = \frac{1 + \operatorname{erf}\left(S_\infty \boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)}{2S_\infty^2}\max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{\left|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right|}, 0\right) + \frac{\sqrt{\pi}}{2S_\infty}\sqrt{\frac{T_{W_j}}{T_\infty}}C_{V_j}$$

being S_∞ the Mach number, T_∞ the atmospheric temperature and T_{W_j} the facet temperature, assumed always equal to 300 K. To handle the jump discontinuities appearing in this model, in the system of equations of motion the Fourier expansions of C_{V_j}, C_{N_j} up to the fourth harmonics are implemented.

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the atmospheric drag force acting of the j-th facet.

The perturbations by the zonal harmonics, the third body acceleration and the Sun gravity are modelled as in Calypso (see Zuiani&Vasile, 2015).

15.1 ATTITUDEPROP_QUAT_GIRF.JL

Propagator for the full coupled attitude and orbital dynamics, with equations of motion expressed in equinoctial elements (a,P1,P2,Q1,Q2,ML), quaternions and components of the angular velocity.

This function handles the inputs, to make them suitable for orbitAttitudeEv_quat (see Section 14), performs the propagation, and performs the required transformation of variables to produce the outputs.

INPUTS:	**civ** = [euler angles 313 [rad], angular velocity [rad/s], a[km], e, i [rad], o[rad],O[rad],nu,[rad]] with a = semi-major axis e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **date0** = Gregorian date at the initial time 0 [year,month,day,hour,minute,seconds] **params** = vector with parameters and settings (see below) *** **tstep** = time-step for the ouput [sec] **tfinal** = propagation time [sec]
OUTPUTS:	**tV** = time vector [sec] with size (nx1) **res** = matrix (nx40) with the outcomes; in each row: • [Jl,Jg,Jh,psil,psig,psih] = sadov variables [kg m^2; rad] • angular velocity in rotating frame [rad/s] • euler angles 313 [rad] • [L,G,H,l,g,h] = andoyer-serret variables [kg m^2; rad] • [a,P1,P2,Q1,Q2,ML] = equinoctial elements • skm=(k/(m+k)) [sadov related variables] • Tk = rotational kinetic energy • Jd = G^2/2*Tk dynamic moment of inertia • savod-like/andoyer-like variables (the code returns sadov-like variables in the case of triaxial satellites or axisymmetric satellites with one moment of inertia equal to the other two – on the contrary, the code returns andoyer-like variables in the case of axisymmetric satellites with equal moments of inertia) • [a,e,I,o,O,nu]

***

params = array of Dictionaries:

[planetsunparams; inertialreframeinfo; satellite; atmosphere; settings]

- **Planetsunparams**
  Dictionary with keys
  - centralBodyIDX **:** integer number identifying the celestial body
    1: Mercury
    2: Venus

3: Earth
4: Mars
5: Jupiter
6: Saturn
7: Uranus
8: Neptune
9: Pluto
10: Sun
11: Moon

- muPlanet = central Planet's gravitational parameter [km^3/s^2]
- rPlanet = Planet radius [km]
- muM = Planet's magnetic dipole strenght [kg km^3/s^2/A]
- psrp = solar radiation pressure [kg/s^2/m] at a reference distance psrp_refdist from the sun
- psrp_refdist = reference distance from the sun for the given value of psrp [km]
- planetRotation = Planet's rotational angular speed [rad/s]
- zonalharmonicscoff = [Ji], i>=2 increasing, Ji zonal harmonics
- perturbingbodiesid = vector containing the ids of the perturbing celestial bodies (third body perturbation)
  1: Mercury
  2: Venus
  3: Earth
  4: Mars
  5: Jupiter
  6: Saturn
  7: Uranus
  8: Neptune
  9: Pluto
  11: Moon
- planet_flat = control integer: it is equal to 1 to consider planet oblateness, equal to 0 otherwise.
- oblateness = oblateness coefficient

NB: if the central body is the Earth, it is sufficient to give as an input just a Dictionary with the key centralBodyIDX and the associated value 3 (planetsunparams=Dict("centralBodyIDX"=>3)). The function will automatically fill the dictionary with the other required fields. In this case, the default value of "planet_flat" is 0.

- **inertialreframeinfo**
  Dictionary with keys
  - ecliptic2inertial = rotation matrix from ecliptic to girf
  - equatorial2inertial = rotation matrix from equatorial to inertial ref frame

  NB: if the central body is the Earth (planetsunparams=Dict("centralBodyIDX"=>3)), the user can give as an input an empty dictionary. In this case, the program assumes that the inertial reference frame is the equatorial reference frame and automatically generated the dictionary.

- **satellite**
  Dictionary with keys

- Moments of Inertia = [A,B,C], principal moment of inertia (A<=B<=C) [Kg m^2]
- intrinsicMagneticMoment = intrinsic magnetic moment of the satellite [A m^2]
- mass = mass of the satellite [kg]
- CD = non-dimensional drag coefficient of the satellite
- numberOfFacets = number of facets in which the satellite external surface is divided
- facets = array of arrays, each associated to a facet of the satellite. In particular,
  facets = [facet_1..facet_k..facet_nf]
  with
  facet_k = [facet_coeff,facet_area,facet_Vinfo_facet_nv]
  where
  facet_coeff = [caj,cdj,csj]
  facet_area = area of the facet
  facet_Vinfo = [rhojv,vv1,vv2,vv3] or [rhojv,vv1,vv2,vv3,vv4]
  rhojv = vector from centre of mass to centroid
  vvi = vertexes of the facet (only triangular and squared facets accepted)

NB: if the input dictionary does not contain the key CD this is automatically introduced with the default value 2.

- **atmosphere**
  Dictionary with keys
  - typeofmodel = integer number identifying the atmospheric model and consequently the drag atmospheric perturbation model to be exploited
    1 : exponential atmospheric model
    2 : nrlmsise00
  - AtmM = matrix containing information required by the atmospheric model

    If typeodmodel = 1
    AtmM = matrix containing the exponential atmospheric model
    in each row: [min altitude [km], density [kg/m^3], scale height [km]

    if typeofmodel = 2
    AtmM = matrix with solar flux data with format
    In each row [jd-AP1-AP2-AP3-AP4-AP5-AP6-AP7-AP8-AP9-AP_AVG-f107_OBS-f107_CENTER81-f107_CENTER90]

    with
    AP1 : Planetary Equivalent Amplitude (Ap) for 0000-0300 UT.
    AP2 : Planetary Equivalent Amplitude (Ap) for 0300-0600 UT.
    AP3 : Planetary Equivalent Amplitude (Ap) for 0600-0900 UT.
    AP4 : Planetary Equivalent Amplitude (Ap) for 0900-1200 UT.
    AP5 : Planetary Equivalent Amplitude (Ap) for 1200-1500 UT.
    AP6 : Planetary Equivalent Amplitude (Ap) for 1500-1800 UT.
    AP7 : Planetary Equivalent Amplitude (Ap) for 1800-2100 UT.
    AP8 : Planetary Equivalent Amplitude (Ap) for 2100-0000 UT.
    AP_AVG : Arithmetic average of the 8 Ap indices for the day.
    F10.7_OBS : Observed 10.7-cm Solar Radio Flux (F10.7). Measured at Ottawa at 1700 UT daily from 1947 Feb 14 until 1991 May 31 and measured at Penticton at 2000 UT from 1991 Jun 01 on. Expressed in units of 10-22 W/m2/Hz.

F10.7_CENTER81 : Centered 81-day arithmetic average of F10.7
F10.7_CENTER90 : Centered 90-day arithmetic average of F10.7.

NB: if the dictionary does not contain the key “typeofmodel” this is automatically added with default value 1 (exponential atmospheric model)

NB: if the central body is the Earth (planetsunparams=Dict(“centralBodyIDX”=>3)), and the user wants to use the exponential atmospheric model, it is sufficient to give as an input a dictionary with the only key “typeofmodel” equal to 1. The key AtmM will be automatically generated, using the exponential model by Vallado (1997), (see Section 7).

- **settings**
  Dictionary with keys
  - includeGravityTorque = Boolean, true if the perturbation by the gravity gradient torque is to be considered
  - includeMagneticTorque = Boolean, true if the perturbation by the residual magnetic torque is to be considered
  - includeSrpTorque = Boolean, true is the perturbation by the light pressure torque is to be considered
  - includeDragTorque = Boolean, true if the perturbation by the atmospheric drag torque is to be considered
  - includeEclipsesEffectsOnAttitude = Boolean, true if the light pressure torque perturbation must be associated with the eclipses effects
  - includeZonalHarmsAcc = Boolean, true is the zonal harmonics perturbation must be considered
  - maxzonalharmonics = natural number equal to the degree of the maximum zonal harmonics to be considered
  - includeThirdBodyAcc = Boolean, true if the third body perturbation must be considered
  - includeSunGravityAcc = Boolean, true if the perturbation by the Sun gravity must be considered
  - includeSrpAcc = Boolean, true if the perturbation by the light pressure acceleration must be considered
  - includeDragAcc = Boolean, true if the perturbation by the atmospheric drag acceleration must be considered
  - includeEclipsesEffectsOnOrbit= Boolean, true if the light pressure acceleration perturbation must be associated with the eclipses effects

This function depends on the functions check_prop_girf and selectreferenceframe_prop_girf. The first is used to check the user’s initial conditions, capturing potential errors and introducing default values. The second is required to perform the transformation to Sadov variables, which are among the outputs.

15.2 ATTITUDEPROP_SADOV_GIRF

Propagator for the full coupled attitude and orbital dynamics, with equations of motion expressed in equinoctial elements (a,P1,P2,Q1,Q2,ML) and Sadov variables

It has the same inputs and outputs of attitudeprop_quat_girf (see Section 15.1).

It is DEPRECATED. Indeed, the equation of motion expressed in Sadov variables for the full dynamics have singularities.

16. SRP_AVERAGE.JL

This file contains functions returning the averaged terms of the equations of motion depending on the light pressure perturbation averaged with respect to the mean anomaly and fast Sadov angles (psil,psig).

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: eclipse.jl; functionlibrary.jl; srp_average.jl

The attitude equations of motion expressed in the variables (skm,Jg,Jh,psil,psig,psih) are

$$\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \mathbf{A}\nabla_{\boldsymbol{s}}\Phi + \mathbf{A}\nabla_{\boldsymbol{s}}\mathcal{V} + \mathbf{B}\boldsymbol{M}, \tag{55}$$

where $\nabla_{\boldsymbol{s}}$ is the gradient operator,

$$\mathbf{A} = \begin{bmatrix} 0 & \tilde{\mathbf{I}} \\ -\tilde{\mathbf{I}}^{\mathrm{T}} & 0 \end{bmatrix}, \tag{56}$$

with

$$\tilde{\mathbf{I}} = \begin{bmatrix} -\frac{\pi}{J_g K(\mu)}\sqrt{\frac{\zeta}{1+\kappa}} & \frac{2(\Pi(-\kappa,\mu)-(1-\zeta)K(\mu))}{J_g K(\mu)} & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \tag{57}$$

and

$$\mathbf{B} = \begin{bmatrix} -\frac{2\zeta}{J_g} b_{13} & -\frac{2\zeta(1-\mu)}{J_g(1+\kappa)} b_{23} & \frac{2(1-\zeta)}{J_g} b_{33} \\ b_{13} & b_{23} & b_{33} \\ \cos\delta b_{13} + \sin\delta b_{12} & \cos\delta b_{23} + \sin\delta b_{22} & \cos\delta b_{33} + \sin\delta b_{32} \\ -\frac{\pi S_x}{2J_g K(\mu)(1-\mu)} & -\frac{\pi S_y}{2J_g K(\mu)} & -\frac{\pi S_z}{2J_g K(\mu)(1-\mu)} \\ \frac{\mathcal{T}S_x}{1-\mu} - \frac{b_{11}\cos\delta}{J_g \sin\delta} & \mathcal{T}S_y - \frac{b_{21}\cos\delta}{J_g \sin\delta} & \frac{\mathcal{T}S_z}{1-\mu} - \frac{b_{31}\cos\delta}{J_g \sin\delta} \\ \frac{b_{11}}{J_g \sin\delta} & \frac{b_{21}}{J_g \sin\delta} & \frac{b_{31}}{J_g \sin\delta} \end{bmatrix},$$

with

$$\mathcal{T} = \frac{(\Pi(-\kappa,\mu) - (1-\zeta)K(m))\sqrt{1+\kappa}}{J_g K(\mu)\sqrt{\zeta}},$$

$$S_x = \frac{\mathrm{dn}(u,\mu)\mathrm{sn}(u,\mu) - \mathrm{cn}(u,\mu)\mathrm{zn}(u,\mu)}{\sqrt{1-\zeta}},$$

$$S_y = \frac{\mathrm{dn}(u,\mu)\mathrm{cn}(u,\mu) + \mathrm{sn}(u,\mu)\mathrm{zn}(u,\mu)}{\sqrt{1+\kappa}\sqrt{1-\zeta}},$$

$$S_z = \frac{\mathrm{dn}(u,\mu)\mathrm{zn}(u,\mu) - \mu\mathrm{cn}(u,\mu)\mathrm{sn}(u,\mu)}{\sqrt{\zeta}},$$

$$b_{11} = -\left(\sqrt{\zeta}\sin(\psi_g+\delta g)\mathrm{cn}(u,\mu)\mathrm{dn}(u,\mu) + \right. \tag{40}$$
$$\left. + \sqrt{1+\kappa}\cos(\psi_g+\delta g)\,\mathrm{sn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$

$$b_{12} = -\left(\sqrt{1+\kappa}\sin(\psi_g+\delta g)\,\mathrm{sn}(u,\mu) + \right. \tag{41}$$
$$\left. - \sqrt{\zeta}\cos(\psi_g+\delta g)\,\mathrm{dn}(u,\mu)\mathrm{cn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$

$$b_{13} = \sqrt{1-\zeta}\,\mathrm{cn}(u,\mu), \tag{42}$$

$$b_{21} = -\left(\cos(\psi_g+\delta g)\,\mathrm{cn}(u,\mu) + \right. \tag{43}$$
$$\left. - \sqrt{1+\kappa}\sqrt{\zeta}\sin(\psi_g+\delta g)\,\mathrm{sn}(u,\mu)\,\mathrm{dn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$

$$b_{22} = -\left(\sqrt{1+\kappa}\sqrt{\zeta}\cos(\psi_g+\delta g)\,\mathrm{sn}(u,\mu)\,\mathrm{dn}(u,\mu) + \right. \tag{44}$$
$$\left. + \sin(\psi_g+\delta g)\,\mathrm{cn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$

$$b_{23} = -\sqrt{1-\zeta}\sqrt{1+\kappa}\,\mathrm{sn}(u,\mu), \tag{45}$$

$$b_{31} = \sqrt{1-\zeta}\sin(\psi_g+\delta g)\,\mathrm{dnk}^{\frac{1}{2}}, \tag{46}$$

$$b_{32} = -\sqrt{1-\zeta}\cos(\psi_g+\delta g)\,\mathrm{dnk}^{\frac{1}{2}}, \tag{47}$$

$$b_{33} = \sqrt{\zeta}\,\mathrm{dn}(u,\mu), \tag{48}$$

where $\mathrm{cn}(u,\mu)$ is the Jacobi elliptic cosine, $\mathrm{sn}(u,\mu)$ is the Jacobi elliptic sine, $\mathrm{dn}(u,\mu)$ is the Jacobi elliptic delta amplitude, and

$$\delta g = -\sqrt{\frac{1+\kappa}{\zeta}}\left(\Pi(-\kappa,\lambda,\mu) - u\frac{\Pi(-\kappa,\mu)}{K(\mu)}\right), \tag{49}$$

$$\mathrm{dnk} = 1 + \kappa\mathrm{sn}^2(u,m), \tag{50}$$

$$u = \frac{2K(\mu)\psi_l}{\pi}. \tag{51}$$

with M the external non-conservative torque, V the potential energy of an external conservative torque. Here, K = ellF(pi/2,m), Π=ellP and ζ=skm.

The light pressure torque is a non-conservative torque. The solar radiation pressure force is modelled as

$$\boldsymbol{f}_{srp} \sim -P_{srp}\left(\frac{1\ au}{|\boldsymbol{r}_{\odot}|}\right)^2 \sum_{j=1}^{n_f} S_j\left(c_{a,j}\boldsymbol{u} + c_{d,j}\boldsymbol{n}_j + c_{s,j}(\boldsymbol{u}\cdot\boldsymbol{n}_j)\boldsymbol{n}_j\right)\left(\frac{1}{3\pi} + \frac{(\boldsymbol{u}\cdot\boldsymbol{n}_j)}{2} + \frac{4(\boldsymbol{u}\cdot\boldsymbol{n}_j)^2}{3\pi}\right)$$

with

P_{srp}	Solar radiation pressure at 1 au
$\boldsymbol{r}_{\odot}$	Sun planetocentric position vector expressed in girf
$\boldsymbol{u}$	It can be se as either ➢ $\frac{\boldsymbol{r}_{\odot}}{\lvert\boldsymbol{r}_{\odot}\rvert}$: approximation of deg 0 ➢ $\frac{\boldsymbol{r}_{\odot}}{\lvert\boldsymbol{r}_{\odot}\rvert}+\frac{(\boldsymbol{r}\cdot\boldsymbol{r}_{\odot})\boldsymbol{r}_{\odot}}{\lvert\boldsymbol{r}_{\odot}\rvert^{3}}-\frac{\boldsymbol{r}}{\lvert\boldsymbol{r}_{\odot}\rvert}$: approximation of deg 1 ➢ $\frac{\boldsymbol{r}_{\odot}}{\lvert\boldsymbol{r}_{\odot}\rvert}+\frac{(\boldsymbol{r}\cdot\boldsymbol{r}_{\odot})\boldsymbol{r}_{\odot}}{\lvert\boldsymbol{r}_{\odot}\rvert^{3}}-\frac{\boldsymbol{r}}{\lvert\boldsymbol{r}_{\odot}\rvert}+\frac{3(\boldsymbol{r}\cdot\boldsymbol{r}_{\odot})^{2}\boldsymbol{r}_{\odot}}{2\lvert\boldsymbol{r}_{\odot}\rvert^{5}}-\frac{r^{2}\boldsymbol{r}_{\odot}}{2\lvert\boldsymbol{r}_{\odot}\rvert^{3}}-\frac{(\boldsymbol{r}\cdot\boldsymbol{r}_{\odot})\boldsymbol{r}}{\lvert\boldsymbol{r}_{\odot}\rvert^{3}}$: approximation of deg 2
$\boldsymbol{n}_j$	Outward normal unit vector of the j-th facet
S_j	Area of the j-th facet
n_f	Number of facets
$c_{a,j}$	$1-\rho_j s_j$, ρ_j total reflectivity of the j-th facet and s_j fraction of ρ_j which is specular
$c_{d,j}$	$\frac{2}{3}(1-s_j)\rho_j$
$c_{s,j}$	$2\rho_j s_j$

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the light pressure force acting of the j-th facet, where $\boldsymbol{\rho}_j$ is the vector from the centre of mass to the centroid of the facet.

Thus, the terms in the equations of motion depending on the fast angles are Mk*bij(skm,psil,psig) and Mk*Sn. Each of this term can be written

Mk*bij/Mk*Sn = dot(VC(alpha),VV(sadov,equinoctial elements)), with VC(alpha) a vector of constant parameters depending on the geometry of the satellite and VV(sadov,equinoctial elements) a vector of coefficients depending on the sadov variables and the equinoctial elements. Furthermore, each component of VV can be expressed as

VV_j = sum f(sadov)*g(equinoctial).

Concerning the orbital equations of motion, in that case the unit vector **u** is set equal to its approximation of deg 0. The solar radiation pressure acceleration is independent of the orbital mean anomaly. Then,

it is averaged over the only Sadov fast angles. Its average can be considered as a constant vector, and the averaged Gauss equations are computed as described in (Zuiani, Vasile, 2015).

NB: in case of axisymmetric satellite with equal principal moments of inertia, the Sadov variables are not well defined. The attitude problem must be expressed in Andoyer variables and only one angle (g) is fast. Thus, the equations of motion are averaged over g and the orbital mean anomaly.

In this file :

Functions	Objective
getAveragedScSunUnitVector (Depending on: getCoeffScSunUnitVector_deg0, getCoeffScSunUnitVector_deg1, getCoeffScSunUnitVector_deg2, getCoeffScSunUnitVector_deg1_corrsrp, getCoeffScSunUnitVector_deg2_corrsrp, averagedterms)	Give the average of the terms depending on the equinoctial elements over the orbital mean anomaly
srpanalyticalaveragedcoeff_T1/2/3, ** srpanalyticalaveragedcoeff_T1/2/3_andoyer_cube	Give the components of the vector VV
srp_initforaverage_T1/2/3	Give the components of the vector VC
srp_averagedterms	Give the averaged terms Mk*bij/Mk*Sn
srpAcc_averaged_psilpsig, srpAcc_averaged_psilpsig_axisym (Depending on: getaveragedterms_psilpsig_srp getaveragedterms_gA_srp)	Give the light pressure acceleration averaged with respect to psil and psig.

** The solar radiation pressure torque is divided into the sum of three terms:

T1—terms of the contribution of the torque depending on the optical coefficients $c_{a,j}$

T2—terms of the contribution of the torque depending on the optical coefficients $c_{d,j}$

T3—terms of the contribution of the torque depending the optical coefficients $c_{s,j}$

NB: At the moment, the unit vector u is set equal to its approximation of deg 0 also for the attitude equations of motion.

NB: there are other functions contained in the file used for the computation of the generating functions (see Section 21).

17. DRAG_AVERAGE.JL

This file contains functions returning the averaged terms of the equations of motion depending on the atmospheric drag perturbation averaged with respect to the mean anomaly and fast Sadov angles (psil,psig).

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra; FastGaussQuadrature; SpecialFunctions; HypergeometricFunctions
- DEPENDENCIES ON INTERNAL FILES: functionlibrary.jl, atmosphere.jl

The attitude equations of motion expressed in the variables (skm,Jg,Jh,psil,psig,psih) are

$$\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \mathbf{A}\nabla_{\boldsymbol{s}}\Phi + \mathbf{A}\nabla_{\boldsymbol{s}}\mathcal{V} + \mathbf{B}\boldsymbol{M}, \tag{55}$$

where $\nabla_{\boldsymbol{s}}$ is the gradient operator,

$$\mathbf{A} = \begin{bmatrix} 0 & \tilde{\mathbf{I}} \\ -\tilde{\mathbf{I}}^{\mathrm{T}} & 0 \end{bmatrix}, \tag{56}$$

with

$$\tilde{\mathbf{I}} = \begin{bmatrix} -\frac{\pi}{J_g K(\mu)}\sqrt{\frac{\zeta}{1+\kappa}} & \frac{2(\Pi(-\kappa,\mu)-(1-\zeta)K(\mu))}{J_g K(\mu)} & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \tag{57}$$

and

$$\mathbf{B} = \begin{bmatrix} -\frac{2\zeta}{J_g} b_{13} & -\frac{2\zeta(1-\mu)}{J_g(1+\kappa)} b_{23} & \frac{2(1-\zeta)}{J_g} b_{33} \\ b_{13} & b_{23} & b_{33} \\ \cos\delta b_{13} + \sin\delta b_{12} & \cos\delta b_{23} + \sin\delta b_{22} & \cos\delta b_{33} + \sin\delta b_{32} \\ -\frac{\pi S_x}{2J_g K(\mu)(1-\mu)} & -\frac{\pi S_y}{2J_g K(\mu)} & -\frac{\pi S_z}{2J_g K(\mu)(1-\mu)} \\ \frac{\mathcal{T} S_x}{1-\mu} - \frac{b_{11}\cos\delta}{J_g \sin\delta} & \mathcal{T} S_y - \frac{b_{21}\cos\delta}{J_g \sin\delta} & \frac{\mathcal{T} S_z}{1-\mu} - \frac{b_{31}\cos\delta}{J_g \sin\delta} \\ \frac{b_{11}}{J_g \sin\delta} & \frac{b_{21}}{J_g \sin\delta} & \frac{b_{31}}{J_g \sin\delta} \end{bmatrix},$$

with

$$\mathcal{T} = \frac{(\Pi(-\kappa,\mu) - (1-\zeta)K(m))\sqrt{1+\kappa}}{J_g K(\mu)\sqrt{\zeta}},$$

$$S_x = \frac{\mathrm{dn}(u,\mu)\mathrm{sn}(u,\mu) - \mathrm{cn}(u,\mu)\mathrm{zn}(u,\mu)}{\sqrt{1-\zeta}},$$

$$S_y = \frac{\mathrm{dn}(u,\mu)\mathrm{cn}(u,\mu) + \mathrm{sn}(u,\mu)\mathrm{zn}(u,\mu)}{\sqrt{1+\kappa}\sqrt{1-\zeta}},$$

$$S_z = \frac{\mathrm{dn}(u,\mu)\mathrm{zn}(u,\mu) - \mu\,\mathrm{cn}(u,\mu)\mathrm{sn}(u,\mu)}{\sqrt{\zeta}},$$

$$b_{11} = -\left(\sqrt{\zeta}\sin\left(\psi_g + \delta g\right)\mathrm{cn}(u,\mu)\mathrm{dn}(u,\mu) + \right. \tag{40}$$
$$\left. + \sqrt{1+\kappa}\cos\left(\psi_g + \delta g\right)\mathrm{sn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$
$$b_{12} = -\left(\sqrt{1+\kappa}\sin\left(\psi_g + \delta g\right)\mathrm{sn}(u,\mu) + \right. \tag{41}$$
$$\left. - \sqrt{\zeta}\cos\left(\psi_g + \delta g\right)\mathrm{dn}(u,\mu)\mathrm{cn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$
$$b_{13} = \sqrt{1-\zeta}\,\mathrm{cn}(u,\mu), \tag{42}$$
$$b_{21} = -\left(\cos\left(\psi_g + \delta g\right)\mathrm{cn}(u,\mu) + \right. \tag{43}$$
$$\left. - \sqrt{1+\kappa}\sqrt{\zeta}\sin\left(\psi_g + \delta g\right)\mathrm{sn}(u,\mu)\,\mathrm{dn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$
$$b_{22} = -\left(\sqrt{1+\kappa}\sqrt{\zeta}\cos\left(\psi_g + \delta g\right)\mathrm{sn}(u,\mu)\,\mathrm{dn}(u,\mu) + \right. \tag{44}$$
$$\left. + \sin\left(\psi_g + \delta g\right)\mathrm{cn}(u,\mu)\right)\mathrm{dnk}^{-\frac{1}{2}},$$
$$b_{23} = -\sqrt{1-\zeta}\sqrt{1+\kappa}\,\mathrm{sn}(u,\mu), \tag{45}$$
$$b_{31} = \sqrt{1-\zeta}\sin\left(\psi_g + \delta g\right)\mathrm{dnk}^{\frac{1}{2}}, \tag{46}$$
$$b_{32} = -\sqrt{1-\zeta}\cos\left(\psi_g + \delta g\right)\mathrm{dnk}^{\frac{1}{2}}, \tag{47}$$
$$b_{33} = \sqrt{\zeta}\,\mathrm{dn}(u,\mu), \tag{48}$$

where $\mathrm{cn}(u,\mu)$ is the Jacobi elliptic cosine, $\mathrm{sn}(u,\mu)$ is the Jacobi elliptic sine, $\mathrm{dn}(u,\mu)$ is the Jacobi elliptic delta amplitude, and

$$\delta g = -\sqrt{\frac{1+\kappa}{\zeta}}\left(\Pi(-\kappa,\lambda,\mu) - u\frac{\Pi(-\kappa,\mu)}{K(\mu)}\right), \tag{49}$$
$$\mathrm{dnk} = 1 + \kappa\mathrm{sn}^2(u,m), \tag{50}$$
$$u = \frac{2K(\mu)\psi_l}{\pi}. \tag{51}$$

with M the external non-conservative torque, V the potential energy of an external conservative torque. Here, K = ellF(pi/2,m), Π=ellP and ζ=skm.

Concerning the atmospheric drag perturbation, two different models are implemented. The first one is based on the exponential model of the atmosphere. It assumes that the drag force is in the direction of the relative incident stream and that the non-dimensional drag coefficient CD is constant, which has to be set by the user as a characteristic of the satellite. The drag force is modelled as

$$\boldsymbol{f}_d = -\frac{1}{2}\rho_\infty v_\infty^2 C_D \sum_{j=1}^{n_f} S_j\left(\frac{1}{3\pi} + \frac{\boldsymbol{n}_j\cdot\boldsymbol{e}_\infty}{2} + \frac{4}{3\pi}\left(\boldsymbol{n}_j\cdot\boldsymbol{e}_\infty\right)^2\right)\boldsymbol{e}_\infty$$

where

ρ_∞	Free stream density
v_∞	Modulus of the aerodynamic flow velocity vector
$\boldsymbol{e}_\infty$	Outward direction of the aerodynamic flow velocity vector
$\boldsymbol{n}_j$	Outward normal unit vector of the j-th facet

S_j	Area of the j-th facet
n_f	Number of facets

The second model is based on the nrlmsise00 model of the atmosphere. The drag force has also a lift component and the non-dimensional drag and lift coefficients depend on the attitude and the Mach number of the rigid body. The drag force is modelled as

$$\boldsymbol{f}_{aero} = \frac{1}{2}\rho_\infty v_\infty^2 \sum_{j=1}^{n_f} S_j (C_{n,j}\boldsymbol{n}_j + C_{v,j}\boldsymbol{e}_\infty)$$

with

$$C_{V_j} = \left(\frac{\exp\left(-S_\infty^2 (\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)^2\right)}{S_\infty \sqrt{\pi}} + \left(1 + \operatorname{erf}\left(S_\infty (\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)\right)\right)(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty) \right) \max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty|}, 0 \right)$$

$$C_{N_j} = \frac{1 + \operatorname{erf}(S_\infty \boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)}{2S_\infty^2} \max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty|}, 0 \right) + \frac{\sqrt{\pi}}{2S_\infty} \sqrt{\frac{T_{W_j}}{T_\infty}} C_{V_j}$$

being S_∞ the Mach number, T_∞ the atmospheric temperature and T_{W_j} the facet temperature, assumed always equal to 300 K. To handle the jump discontinuities appearing in this model, in the system of equations of motion the Fourier expansions of C_{V_j}, C_{N_j} up to the fourth harmonics are implemented.

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the atmospheric drag force acting of the j-th facet, where $\boldsymbol{\rho}_j$ is the vector from the centre of mass to the centroid of the facet.

The terms in the equations of motion depending on the fast angles are Mk*bij(skm,psil,psig) and Mk*Sn. Each of this term can be written

Mk*bij/Mk*Sn = dot(VC(alpha),VV(sadov,equinoctial elements)),

with VC(alpha) a vector of constant parameters depending on the geometry of the satellite and VV(sadov,equinoctial elements) a vector of coefficients depending on the sadov variables and the equinoctial elements. Furthermore, each component of VV can be expressed as

VV_j = sum f(sadov)*g(equinoctial).

Similarly, the planetary gauss equation contains terms ak*gi(M), with ak the component of the perturbing acceleration and gi some function of the orbital mean anomaly. Each term can be written as

ak*gi = KC(beta)*KK(sadov,equinoctial)

with KC(beta) a vector of constant parameters and KK(sadov,equinoctial) a vector of coefficients depending on the sadov variables and the equinoctial elements. Furthermore, each component of VV can be expressed as

KK_j = sum h(sadov)*d(equinoctial).

NB: in case of axisymmetric satellite with equal principal moments of inertia, the Sadov variables are not well defined. The attitude problem must be expressed in Andoyer variables and only one angle (g) is fast. Thus, the equations of motion are averaged over g and the orbital mean anomaly.

In this file :

Functions	Objective
dragnumericallyaveragedterms (Depending on: organizeMAterms_dragforce getMAterms_drag)	Perform the numerical average of the terms depending on the equinoctial elements over the orbital mean anomaly
dragsemianalyticalaveragedcoeff_T1/2/3/4 dragsemianalyticalaveragedcoeff_T1/2/3/4_andoyer_cube ** (Depending on dragsemianalyticalaveragedcoeff_fN_O3 dragsemianalyticalaveragedcoeff_fN_O4 dragsemianalyticalaveragedcoeff_fV_O3 dragsemianalyticalaveragedcoeff_fV_O4 dragsemianalyticalaveragedcoeff_fN_O3_andoyer_cube dragsemianalyticalaveragedcoeff_fN_O4_andoyer_cube dragsemianalyticalaveragedcoeff_fV_O3_andoyer_cube dragsemianalyticalaveragedcoeff_fV_O4_andoyer_cube integrals_bijO6 integrals_bijSiO6)	Give the components of the vector VV
drag_initforaverage_T1/2/3/4	Give the components of the vector VC
Getdragsaaveragedtermsattitudeeq (Depending on dragsemianalyticalaveragedcoeff_T1/2/3/4 dragsemianalyticalaveragedcoeff_T1/2/3/4_andoyer_cube drag_averagedterms)	Give the averaged terms to be introduced in the equations of motion
drag_initforaverage_TV/N	Give the components of the vector KC
getdragaveragedtermsgausseq_strategy1 getdragaveragedtermsgausseq_strategy1_cube (Depending on: getDragForceTermsAveragedWRTAttitude getDragForceTermsAveragedWRTAttitude_cube)	Give the averaged terms to be introduced in the planetary Gauss equations.

** T1,T2,T3,T4 refer to the subdivision of the atmospheric drag torque is divided into the sum of different terms

NB: there are other functions contained in the file used for the computation of the generating functions (see Section 21).

18. AVERAGEDORBITPERT.JL

This file contains function returning the averaged planetary Gauss equations.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: srp_average.jl; drag_average.jl

In this file :

Functions	Perturbation
Zonalharmonicsaveragedgauss (Depending on getj2averagedterms getj3averagedterms getj4averagedterms getj5averagedterms)	Zonal harmonics
tbaveragedgauss_wrapper1 (Depending on tbaveragedgauss getaveragedterms_3bp_deg2 getaveragedterms_3bp_deg3 getaveragedterms_3bp_deg4 getaveragedterms_3bp_deg5 getaveragedterms_3bp_deg6)	Third body perturbation
srpaveragedgauss_wrapper1_AV (for triaxial satellites) srpaveragedgauss_wrapper2_AV (for axisymmetric satellites with equal principal moments of inertia) (Depending on srpaveragedgauss_wrapper1_AV_OURMODEL srpaveragedgauss_wrapper2_AV_OURMODEL srpaveragedgauss getdefintegrals_0_EL_forconstantaccelerationininertialframeaveraged getconstantaccelerationininertialframeaveragedterms)	Light pressure
dragveragedgauss (for triaxial satellites) dragveragedgauss_cube (for axisymmetric satellites with equal principal moments of inertia)	Atmospheric drag

19. SAM_HIGHERORDERTERMS.JL

By means of the Lie transformations approach it is possible to compute higher order averaged terms to be included in the average model to increase its accuracy. This file contains higher order terms computed for the attitude averaged equations of motion for the gravity-gradient and residual magnetic torques perturbation when the satellite is triaxial or axysimmetric (but *not* if all the three principal moments of inertia are equal). It contains a function gethigherordercorrections (which depends on several other functions present in the file) returning the higher order correction to be included in the average equations of motion.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: -

20. EQMOTION_AVERAGEDMODEL_GIRF.JL

This file contains the function returning the system of ordinary differential equations describing the avreged attitude-orbital dynamics of a rigid body orbiting a planet.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: srp_averaged.jl; drag:_averaged.jl; averagedorbitpert.jl; sam_higherorderterms.jl

The environmental perturbations considered are:

- Gravity-gradient torque
- Residual magnetic torque
- Light pressure torque
- Atmospheric drag torque
- Zonal harmonics J2, J3, J4, J5
- Third body acceleration (ex. Lunar gravity for a body orbiting the Earth)
- Sun gravity
- Light pressure acceleration
- Atmospheric drag acceleration

Concerning the atmospheric drag perturbation, two different models are implemented, one associated to the exponential model of the atmosphere and one associated to the nrlmsise00 model.

For further details see next Section semianalyticalpropagators_girf.jl

20.1 ATTITUDEMEANEV_SEMIANALYTICAL_TRIAXIAL

This function returns the ODE describing the averaged coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are the modified Sadov variables (skm,Jg,Jh,psil,psig,psih). This ODE system is suitable only for triaxial satellites or axisymmetric satellites with one moment of inertia different from the other two.

Consider an inertial reference framx XYZ and a rotating reference frame xyz co-moving with the satellite. The body reference frame xyz selected must be such that that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.

delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [skm,Jg,Jh,psil,psig,psih,a,P1,P2,Q1,Q2,ML] where skm = k/(m+k) Jg = G Jh = G*cos(delta), psil = pi/2 ellF(pi/2,m)ellF(lambda,m), psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), psih = angle between X axis and N a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), ML = mean longitude with e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination delta = inclination angle between the XY plane and the plane perpendicular to GV. lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis g = angle between N and N'' ** ellF, ellP elliptic integrals (see Section 4) **p** = array with parameters and settings (*** see Section 22) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the functions (included in the file): gravitytorque_girf_triaxial, magnetictorque_girf_triaxial, srptorque_semianalytical, dragtorque_semianalytical, getsingularityterms

20.2 ATTITUDEMEANEV_SEMIANALYTICAL_TRIAXIAL_SADOVLIKE

This function returns the ODE describing the averaged coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are the Sadov-like variables (J1,J2,J3,J4,J5,J6,J7). This ODE system is suitable

only for triaxial satellites or axisymmetric satellites with one moment of inertia different from the other two.

Consider an inertial reference framx XYZ and a rotating reference frame xyz co-moving with the satellite. The body reference frame xyz selected must be such that that m belongs to [0,1] and k>0, where
k = C/A*(B-A)/(C-B)
m = k*(C-delta)/(delta-A)
with
A,B,C = principal moments of inertia, A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
delta = G^2/2/Phi, with G the magnitude of the angular momentum of the body and Phi the kinetic energy
Let N' be the node between the XY and xy planes.
Let GV be angular momentum of the body (G = norm(GV)).
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [J1,J2,J3,J4,J5,J6,J7,a,P1,P2,Q1,Q2,ML] where J1 = skm J2 = Jg J3 = Jh J4 = psil J5 = psig + Jh/Jg*psih J6 = sqrt(Jg^2-Jh^2)*cos(psih) J7 = sqrt(Jg^2-Jh^2)*sin(psih) a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), ML = mean longitude with e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination skm = k/(m+k) Jg = G Jh = G*cos(delta), psil = pi/2 ellF(pi/2,m)ellF(lambda,m), psig = g - sqrt{(k+m)(1+k)/k}(ellP(-k,pi/2,m)ellF(lambda,m)/ellF(pi/2,m)-ellP(-k,lambda,m)), psih = angle between X axis and N delta = inclination angle between the XY plane and the plane perpendicular to GV.

	lambda = atan(cos(l),sqrt{1+k}sin(l)), l = angle between N'' and x axis g = angle between N and N'' ** ellF, ellP elliptic integrals (see Section 4) **p** = array with parameters and settings (*** see Section 22) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the functions (included in the file): gravitytorque_girf_triaxial_sadovlike, magnetictorque_girf_triaxial_sadovlike, srptorque_semianalytical_sadovlike, dragtorque_semianalytical_sadovlike, getsingularityterms_sadovlike

20.3 ATTITUDEMEANEV_SEMIANALYTICAL_AXISYMM

This function returns the ODE describing the averaged coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are the Andoyer-Serret variables (L,G,H,l,g,h). This ODE system is suitable only for axisymmetric satellites with equal principal moments of inertia.

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV =[p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [L,G,H,l,g,h,a,P1,P2,Q1,Q2,ML] where L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV| H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and Na = semi-major axis a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), ML = mean longitude with

	e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **p** = array with parameters and settings (*** see Section 22) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the function (included in this file): magnetictorque_girf_axisym, srptorque_semianalytical_axisymm, dragtorque_semianalytical_axisymm

20.3 ATTITUDEMEANEV_SEMIANALYTICAL_AXISYMM_ANODYERLIKE

This function returns the ODE describing the averaged coupled attitude-orbital dynamics of a rigid body. The orbital variables used are the equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables used are the Andoyer-like variables (J1,J2,J3,J4,J5,J6,J7). This ODE system is suitable only for axisymmetric satellites with equal principal moments of inertia.

Consider an inertial reference Frame XYZ.
Consider a body reference frame xyz in principal axes of inertia.
Let A,B,C be the principal moments of inertia with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.
Let GV=diag([A,B,C])omV be angular momentum of the body (omV =[p,q,r]^T is the angular velocity)
Let N be the node between the inertial reference XY plane and the plane perpendicular to GV; let N' be the node between the XY and xy planes; let N'' be the node between the plane perpendicular to GV and the xy plane.

INPUTS:	**du** = allocated vector with the same dimension of the output **u** = [L,G,H,l,g,h,a,P1,P2,Q1,Q2,ML] where J1 = L J2 = G J3 = H J4 = l J5 = g + H/G*h J6 = sqrt(G^2-H^2)*cos(h) J7 = sqrt(G^2-H^2)*sin(h) a = semi-major axis P1 = e*sin(O+o) P2 = e*cos(O+o) Q1 = tan(i/2)*sin(OM), Q2 = tan(i/2)*cos(OM), ML = mean longitude With L = |GV|cos(sigma), sigma = inclination angle between the plane perpendicular to GV and the xy plane G = |GV|

	H = |GV|cos(delta), delta = inclination angle between the XY plane and the plane perpendicular to GV. l = angle between N'' and x axis g = angle between N and N'' h = angle between X axis and N e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **p** = array with parameters and settings (*** see Section 22) **t** = time
OUTPUTS:	**du** = vector with the time derivatives of the state vector

This function depends on the function (included in this file): magnetictorque_girf_axisym_andoyerlike, srptorque_semianalytical_axisymm_andoyerlike, dragtorque_semianalytical_axisymm_andoyerlike

21. GENERATINGFUNCTIONS

There are several files which contains functions used to compute the generating functions of the Lie transformation from mean to osculating attitude variables and viceversa. Given the modified Sadov variablas s=(skm,Jg,Jh,psil,psig,psih) or the sadov-like variables s, the transformation from osculating to mean variables is given by

$$\bar{\boldsymbol{s}} = \boldsymbol{s} - \boldsymbol{W}_2(\boldsymbol{s}) - \boldsymbol{W}_{2,bis}\big(\boldsymbol{s} - \boldsymbol{W}_2(\boldsymbol{s})\big) - \boldsymbol{W}_3(\boldsymbol{s} - \boldsymbol{W}_2(\boldsymbol{s}) - \boldsymbol{W}_{2,bis}\big(\boldsymbol{s} - \boldsymbol{W}_2(\boldsymbol{s})\big))$$

The functions return W2,W2bis and W3.

In particular:

- generatingfunctiongravity.jl contains the functions returning the generating functions for the gravity perturbations:
  - getWG2 returns the value of W2 for the modified sadov variables
  - getWG2sadovlike returns the value of W2 for the sadov-like variables
  - getWG3 returns the value of W3 for the modified sadov variables
  - getWG3sadovlike returns the value of W3 for the sadov-like variables

  They depend on several other functions contained in the file and in the file generatingfunctionsterms_casesmk0.jl

- generatingfunctionmagnetic.jl contains the functions returning the generating functions for the magnetic perturbations:
  - getWM2 returns the value of W2 for the modified sadov variables
  - getWM2sadovlike returns the value of W2 for the sadov-like variables
  - getWM3 returns the value of W3 for the modified sadov variables
  - getWM3sadovlike returns the value of W3 for the sadov-like variables

  They depend on several other functions contained in the file and in the file generatingfunctionsterms_casesmk0.jl

- generatingfunctionsrpdrag_p.jl 1 contains the functions returning the generating functions for the light pressure perturbations:
  - getWSRP2 returns the value of W2 for the modified sadov variables

- getWSRP2sadovlike returns the value of W2 for the sadov-like variables
- getWDRAG2 returns the value of W2 for the modified sadov variables
- getWDRAG2sadovlike returns the value of W2 for the sadov-like variables

They depend on several other perturbations contained in the file, in the other files generatingfunction_srp_drag_termsP1.jl, generatingfunction_srp_drag_termsP2.jl, generatingfunction_srp_drag_termsP3.jl and on the file srp_average.jl, drag_average.jl and generatingfunctionsterms_casesmk0.jl

- generatingfunctionsrp_p2.jl contains the functions returning the generating functions for the srp perturbations:
  - getWSRP2bis returns the value of W2bis for the modified sadov variables
  - getWSRP2bissadovlike returns the value of W2bis for the sadov-like variables
  - getWSRP3 returns the value of W3 for the modified sadov variables
  - getWSRP3sadovlike returns the value of W3 for the sadov-like variables

  This functions depend on several other functions contained in the file, and on srp_average.jl

- generatingfunctiondrag_p2.jl contains the functions returning the generating functions for the drag perturbations:
  - getWDRAG2bis returns the value of W2bis for the modified sadov variables
  - getWDRAG2bissadovlike returns the value of W2bis for the sadov-like variables
  - getWDRAG3 returns the value of W3 for the modified sadov variables
  - getWDRAG3sadovlike returns the value of W3 for the sadov-like variables

  This functions depend on several other functions contained in the file, and on drag_average.jl

22. SEMIANALYTICALPROPAGATORS_GIRF

This file contains the propagator of the averaged coupled orbital and attitude dynamics.

- DEPENDENCIES ON EXTERNAL PACKAGES: LinearaAlgebra;
- DEPENDENCIES ON INTERNAL FILES: astroConstants.jl; orbitvarchange.jl; attitudevarchange.j; eqMotion_averagedmodel_girf.jl; srp_averag.jl; drag_average.jl; generatingfunctiongravity.jl; generatingfunctionmagnetic.jl; generatingfunctionsrpdrag_p1; generatingfunctionsrp_p2.jl; generatingfunctiondrag_p2.jl.

Two reference frame are considered. An generic inertial reference frame girf and a rotating reference frame Oxyz co-moving with the satellite. The frame Oxyz **must** be centred in the satellite centre of mass and must be in principal axes, such that the principal moments of inertia are A<=B<=C, with A=int(y^2+z^2)dm, B=int(x^2+z^2)dm and C=int(x^2+y^2)dm.

The environmental perturbations considered are:

- Gravity-gradient torque
- Residual magnetic torque
- Light pressure torque
- Atmospheric drag torque
- Zonal harmonics J2, J3, J4, J5
- Third body acceleration (ex. Lunar gravity for a body orbiting the Earth)
- Sun gravity
- Light pressure acceleration

- Atmospheric drag acceleration

The gravity-gradient torque is modelled as

$$\boldsymbol{M}_g = \frac{3\mu_P}{r^3}\begin{bmatrix}(C-B)\alpha_2\alpha_3\\(A-C)\alpha_3\alpha_1\\(B-A)\alpha_1\alpha_2\end{bmatrix}$$

where r is the planetocentric distance of the satellite's centre of mass, μ_P is the planet's gravitational parameter and $(\alpha_1, \alpha_2, \alpha_3)$ are the direction cosines of the position vector of the satellite's centre of mass expressed in Oxyz.

The magnetic torque is modelled as

$$M_m = I \times R_{i2b}B$$

Where I is the intrinsic magnetic moment of the satellite (expressed in Oxyz), R_{i2b} is the rotation matrix from girf to Oxyz and

$$B = \frac{\mu_m}{r^3}\left(\boldsymbol{e}_z - \frac{3\left(\boldsymbol{e}_Z \cdot \frac{\boldsymbol{r}}{r}\right)\boldsymbol{r}}{r}\right)$$

where μ_m is the magnetic dipole strength of the planet, $\boldsymbol{e}_z$ is the unit vector in the direction of the magnetic field dipole (expressed in girf) and $\boldsymbol{r}$ is the position vector of the satellite centre of mass (expressed in girf).

NB: the propagator is thought for satellites orbiting the Earth, so $\boldsymbol{e}_z$ is approximated as the unit vector in the direction of the Earth's north pole. The code may need to be corrected if this assumption does not hold for a different central body.

Both the light pressure and the atmospheric drag perturbations are based on a subdivision of the external surface of the rigid body in either triangular or squared facets. The total perturbation is given by the sum of the perturbations acting on each facets.

The light pressure force is

$$\boldsymbol{f}_{srp} \sim -P_{srp}\left(\frac{1\ au}{|\boldsymbol{r}_\odot|}\right)^2 \sum_{j=1}^{n_f} S_j\left(c_{a,j}\boldsymbol{u} + c_{d,j}\boldsymbol{n}_j + c_{s,j}(\boldsymbol{u}\cdot\boldsymbol{n}_j)\boldsymbol{n}_j\right)\left(\frac{1}{3\pi} + \frac{(\boldsymbol{u}\cdot\boldsymbol{n}_j)}{2} + \frac{4(\boldsymbol{u}\cdot\boldsymbol{n}_j)^2}{3\pi}\right)$$

with

P_{srp}	Solar radiation pressure at 1 au
$\boldsymbol{r}_\odot$	Sun planetocentric position vector expressed in girf
$\boldsymbol{u}$	$\frac{\boldsymbol{r}_\odot}{\lvert\boldsymbol{r}_\odot\rvert}$
$\boldsymbol{n}_j$	Outward normal unit vector of the j-th facet

S_j	Area of the j-th facet
n_f	Number of facets
$c_{a,j}$	$1 - \rho_j s_j$, ρ_j total reflectivity of the j-th facet and s_j fraction of ρ_j which is specular
$c_{d,j}$	$\frac{2}{3}(1 - s_j)\rho_j$
$c_{s,j}$	$2\rho_j s_j$

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the light pressure force acting of the j-th facet, where $\boldsymbol{\rho}_j$ is the vector from the centre of mass to the centroid of the facet.

Concerning the atmospheric drag perturbation, two different models are implemented. The first one is based on the exponential model of the atmosphere. It assumes that the drag force is in the direction of the relative incident stream and that the non-dimensional drag coefficient CD is constant, which has to be set by the user as a characteristic of the satellite. The drag force is modelled as

$$\boldsymbol{f}_d = -\frac{1}{2}\rho_\infty v_\infty^2 C_D \sum_{j=1}^{n_f} S_j \left(\frac{1}{3\pi} + \frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{2} + \frac{4}{3\pi}\left(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty\right)^2\right)\boldsymbol{e}_\infty$$

where

ρ_∞	Free stream density
v_∞	Modulus of the aerodynamic flow velocity vector
$\boldsymbol{e}_\infty$	Outward direction of the aerodynamic flow velocity vector

The second model is based on the nrlmsise00 model of the atmosphere. The drag force has also a lift component and the non-dimensional drag and lift coefficients depend on the attitude and the Mach number of the rigid body. The drag force is modelled as

$$\boldsymbol{f}_{aero} = \frac{1}{2}\rho_\infty v_\infty^2 \sum_{j=1}^{n_f} S_j (C_{n,j}\boldsymbol{n}_j + C_{v,j}\boldsymbol{e}_\infty)$$

with

$$C_{V_j} = \left(\frac{\exp\left(-S_\infty^2 (\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)^2\right)}{S_\infty \sqrt{\pi}} + \left(1 + \operatorname{erf}\left(S_\infty (\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)\right)\right)(\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty) \right) \max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty|}, 0 \right)$$

$$C_{N_j} = \frac{1 + \operatorname{erf}(S_\infty \boldsymbol{n}_j \cdot \boldsymbol{e}_\infty)}{2S_\infty^2} \max\left(\frac{\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty}{|\boldsymbol{n}_j \cdot \boldsymbol{e}_\infty|}, 0 \right) + \frac{\sqrt{\pi}}{2S_\infty} \sqrt{\frac{T_{W_j}}{T_\infty}} C_{V_j}$$

being S_∞ the Mach number, T_∞ the atmospheric temperature and T_{W_j} the facet temperature, assumed always equal to 300 K. To handle the jump discontinuities appearing in this model, in the system of equations of motion the Fourier expansions of C_{V_j}, C_{N_j} up to the fourth harmonics are implemented.

The torque is coherently computed by summing the contribution of each facet, computed as the cross product between $\boldsymbol{\rho}_j$ and the atmospheric drag force acting of the j-th facet.

The perturbations by the zonal harmonics, the third body acceleration and the Sun gravity are modelled as in Calypso (see Zuiani&Vasile, 2015).

22.1 ATTITUDEAVERAGEDPROP_SEMINALYTICAL_GIRF

Propagator for the averaged coupled attitude and orbital dynamics. The equations of motion expressed in equinoctial elements (a,P1,P2,Q1,Q2,ML). The attitude variables are automatically selected by the propagator among modified sadov variables (skm,Jg,Jh,psil,psig,psih), sadov-like variables, andoyer and andoyer-like variables depending on the geometry of the satellite and the initial conditions.

This function handles the inputs, selects the suitable variables to be used for the propagation, performs the conversion from osculating to mean variables, integrates the averaged ODE system (see eqmotion_averagedmodel_girf.jl) depending on the selected variables, and performs the required transformation of variables to produce the outputs.

The outcomes of the propagators are valid only for a non-chaotic dynamics. If the initial state falls inside the chaotic region or it is very close to it, the averaged attitude equations are not suitable.

INPUTS:	**civ** = [euler angles 313 [rad], angular velocity [rad/s], a[km], e, i [rad], o[rad],O[rad],nu,[rad]] with a = semi-major axis e = eccentricity O = longitude of ascending node o = argument of pericentre i = inclination **date0** = Gregorian date at the initial time 0 **params** = vector with parameters and settings (see below) *** **tstep** = time-step for the ouput [sec] **tfinal** = propagation time [sec]
OUTPUTS:	**tV** = time vector [sec] with size (nx1) **res** = matrix (nx40) with the outcomes; in each row: • [Jl,Jg,Jh,psil,psig,psih] = sadov variables [kg m^2; rad] • angular velocity in rotating frame [rad/s] • euler angles 313 [rad]

	• [L,G,H,l,g,h] = andoyer-serret variables [kg m^2; rad] • [a,P1,P2,Q1,Q2,ML] = equinoctial elements • skm=(k/(m+k)) [sadov related variables] • Tk = rotational kinetic energy • Jd = G^2/2*Tk dynamic moment of inertia • savod-like/andoyer-like variables (the code returns sadov-like variables in the case of triaxial satellites or axisymmetric satellites with one moment of inertia equal to the other two – on the contrary, the code returns andoyer-like variables in the case of axisymmetric satellites with equal moments of inertia) [a,e,I,o,O,nu]

***

params = array of Dictionaries:

[planetsunparams; inertialrefframeinfo; satellite; atmosphere; settings]

- **Planetsunparams**
  Dictionary with keys
  - centralBodyIDX **:** integer number identifying the celestial body
    1: Mercury
    2: Venus
    3: Earth
    4: Mars
    5: Jupiter
    6: Saturn
    7: Uranus
    8: Neptune
    9: Pluto
    10: Sun
    11: Moon
  - muPlanet = central Planet's gravitational parameter [km^3/s^2]
  - rPlanet = Planet radius [km]
  - muM = Planet's magnetic dipole strenght [kg km^3/s^2/A]
  - psrp = solar radiation pressure [kg/s^2/m] at a reference distance psrp_refdist from the sun
  - psrp_refdist = reference distance from the sun for the given value of psrp [km]
  - planetRotation = Planet's rotational angular speed [rad/s]
  - zonalharmonicscoff = [Ji], i>=2 increasing, Ji zonal harmonics
  - perturbingbodiesid = vector containing the ids of the perturbing celestial bodies (third body perturbation)
    1: Mercury
    2: Venus
    3: Earth
    4: Mars
    5: Jupiter
    6: Saturn

7: Uranus
8: Neptune
9: Pluto
11: Moon

- planet_flat = control integer: it is equal to 1 to consider planet oblateness, equal to 0 otherwise.
- oblateness = oblateness coefficient

NB: if the central body is the Earth, it is sufficient to give as an input just a Dictionary with the key centralBodyIDX and the associated value 3 (planetsunparams=Dict("centralBodyIDX"=>3)). The function will automatically fill the dictionary with the other required fields. In this case, the default value of "planet_flat" is 0.

- **inertialrefframeinfo**
  Dictionary with keys
  - ecliptic2inertial = rotation matrix from ecliptic to girf
  - equatorial2inertial = rotation matrix from equatorial to inertial ref frame

  NB: if the central body is the Earth (planetsunparams=Dict("centralBodyIDX"=>3)), the user can give as an input an empty dictionary. In this case, the program assumes that the inertial reference frame is the equatorial reference frame and automatically generated the dictionary.

- **satellite**
  Dictionary with keys
  - Moments of Inertia = [A,B,C], principal moment of inertia (A<=B<=C) [Kg m^2]
  - intrinsicMagneticMoment = intrinsic magnetic moment of the satellite [A m^2]
  - mass = mass of the satellite [kg]
  - CD = non-dimensional drag coefficient of the satellite
  - numberOfFacets = number of facets in which the satellite external surface is divided
  - facets = array of arrays, each associated to a facet of the satellite. In particular,
    facets = [facet_1..facet_k..facet_nf]
    with
    facet_k = [facet_coeff,facet_area,facet_Vinfo_facet_nv]
    where
    facet_coeff = [caj,cdj,csj]
    facet_area = area of the facet
    facet_Vinfo = [rhojv,vv1,vv2,vv3] or [rhojv,vv1,vv2,vv3,vv4]
    rhojv = vector from centre of mass to centroid
    vvi = vertexes of the facet (only triangular and squared facets accepted)

NB: if the input dictionary does not contain the key CD this is automatically introduced with the default value 2.

- **atmosphere**
  Dictionary with keys
  - typeofmodel = integer number identifying the atmospheric model and consequently the drag atmospheric perturbation model to be exploited
    1 : exponential atmospheric model
    2 : nrlmsise00
  - AtmM = matrix containing information required by the atmospheric model

If typeodmodel = 1
AtmM = matrix containing the exponential atmospheric model
in each row: [min altitude [km], density [kg/m^3], scale height [km]

if typeofmodel = 2
AtmM = matrix with solar flux data with format
In each row [jd-AP1-AP2-AP3-AP4-AP5-AP6-AP7-AP8-AP9-AP_AVG-f107_OBS-f107_CENTER81-f107_CENTER90]

with

AP1 : Planetary Equivalent Amplitude (Ap) for 0000-0300 UT.
AP2 : Planetary Equivalent Amplitude (Ap) for 0300-0600 UT.
AP3 : Planetary Equivalent Amplitude (Ap) for 0600-0900 UT.
AP4 : Planetary Equivalent Amplitude (Ap) for 0900-1200 UT.
AP5 : Planetary Equivalent Amplitude (Ap) for 1200-1500 UT.
AP6 : Planetary Equivalent Amplitude (Ap) for 1500-1800 UT.
AP7 : Planetary Equivalent Amplitude (Ap) for 1800-2100 UT.
AP8 : Planetary Equivalent Amplitude (Ap) for 2100-0000 UT.
AP_AVG : Arithmetic average of the 8 Ap indices for the day.
F10.7_OBS : Observed 10.7-cm Solar Radio Flux (F10.7). Measured at Ottawa at 1700 UT daily from 1947 Feb 14 until 1991 May 31 and measured at Penticton at 2000 UT from 1991 Jun 01 on. Expressed in units of 10-22 W/m2/Hz.
F10.7_CENTER81 : Centered 81-day arithmetic average of F10.7
F10.7_CENTER90 : Centered 90-day arithmetic average of F10.7.

- considerthermalCD = boolean (if false in the drag model the friction effects are neglected)

NB: if the dictionary does not contain the key “typeofmodel” this is automatically added with default value 1 (exponential atmospheric model)

NB: if the central body is the Earth (planetsunparams=Dict(“centralBodyIDX”=>3)), and the user wants to use the exponential atmospheric model, it is sufficient to give as an input a dictionary with the only key “typeofmodel” equal to 1. The key AtmM will be automatically generated, using the exponential model by Vallado (1997), (see Section 7).

NB: if the key considerthermalCD is missing, it is automatically set with the default value “true”.

- **settings**
  Dictionary with keys
  - includeGravityTorque = Boolean, true if the perturbation by the gravity gradient torque is to be considered
  - includeMagneticTorque = Boolean, true if the perturbation by the residual magnetic torque is to be considered
  - includeSrpTorque = Boolean, true is the perturbation by the light pressure torque is to be considered

- includeDragTorque = Boolean, true if the perturbation by the atmospheric drag torque is to be considered
- includeEclipsesEffectsOnAttitude = Boolean, true if the light pressure torque perturbation must be associated with the eclipses effects
- includeZonalHarmsAcc = Boolean, true is the zonal harmonics perturbation must be considered
- maxzonalharmonics = natural number equal to the degree of the maximum zonal harmonics to be considered
- includeThirdBodyAcc = Boolean, true if the third body perturbation must be considered
- includeSunGravityAcc = Boolean, true if the perturbation by the Sun gravity must be considered
- includeSrpAcc = Boolean, true if the perturbation by the light pressure acceleration must be considered
- includeDragAcc = Boolean, true if the perturbation by the atmospheric drag acceleration must be considered
- includeEclipsesEffectsOnOrbit= Boolean, true if the light pressure acceleration perturbation must be associated with the eclipses effects
- checkchaoticity = Boolean, true if the user wants to check during the propagation the proximity to the chaotic region. If close to the chaotic region the semi-analytical propagation is interrupted.
- Higherordercorrections = Boolena, true if the user wants to include higher order terms in the averaged attitude equations of motion
- osculatingpropagationinchaoticregion = Boolean: if checkchaoticity = true and osculatingpropagationinchaoticregion= true in the chaotic region the propagation of the dynamics is performed by the osculating propagator.
- sunfixed = Boolean, if true the position of the Sun is assumed fixed in the space
- sunposition = position of the Sun in space if sunfixed=true

This function depends on the internal functions:

- checksandhandleinputs, handleinputs, selectreferenceframe_girf used to check the user's input, introduce the default values, and handle the input making them suitable to be used to integrate the ODE system.
- transformationci_keplerian to convert the osculating equinoctial elements into mean equinoctial elements
- transformationci_triaxial, transformationci_triaxial_sadovlike to convert the osculating modified sadov variables (or sadov-like variables in mean modified sadov variables (or sadov-like variables), semi-analytically, on the basis of the Lie transformations approach
- transformationci_triaxial_numerical, transformationci_triaxial_sadovlike_numerical to convert the osculating modified sadov variables (or sadov-like variables in mean modified sadov variables (or sadov-like variables), using a numerical procedure [NOT USED]
- transformationci_axysimmetric_numerical, transformationci_axysimmetric_andoyerlike to convert the osculating Andoyer-Serret variables (or andoyer-like variables in mean Andoyer-Serret variables (or andoyer-like variables), using a numerical procedure

- prop_semianalytical_st_girf_triax, prop_semianalytical_st_girf_triax_sadovlike, prop_semianalytical_st_girf_axisymm, prop_semianalytical_st_girf_axisymm_andoyerlike: actual propagators, used depending on the type of attitude variables selected.