SatStress

API Documentation

March 28, 2008

Contents

1 Package satstress

Tools for analysing the relationship between tidal stresses and tectonics on icy satellites.

Written by Zane Selvans[1] (`zane.selvans@colorado.edu`[2]) as part of his Ph.D. dissertation research.

The project is hosted at Google Code: *http://code.google.com/p/satstress*

(section) 1 Installation

Hopefully getting `SatStress` to work on your system is a relatively painless process, however, the software does assume you have basic experience with the Unix shell and programming within a Unix environment (though it should work on Windows too). In particular, this installation information assumes you already have and are able to use:

- compilers for both C and Fortran. Development has been done on Mac OS X (10.5) using the GNU compilers `gcc` and `g77`, so those should definitely work. On other systems, with other compilers, your mileage may vary.
- the `make` utility, which manages dependencies between files.

(section) 1.1 Other Required and Recommended Software

To get the `SatStress` package working, you'll need to install some other (free) software first:

- **Python 2.5** or later (*http://www.python.org*). If you're running a recent install of Linux, or Apple's Leopard operating system (OS X 10.5.x), you already have this. Python is also available for Microsoft Windows, and just about any other platform you can think of.
- **SciPy** (*http://www.scipy.org*), a collection of scientific libraries that extend the capabilities of the Python language.

In addition, if you want to use `GridCalc`, you'll need:

- **netCDF** (*http://www.unidata.ucar.edu/software/netcdf/*, a library of routines for storing, retrieving, and annotating regularly gridded multi-dimensional datasets. Developed by Unidata[3]
- **netcdf4-python** (*http://code.google.com/p/netcdf4-python/*, a Python interface to the netCDF library.

If you want to actually view `GridCalc` output, you'll need a netCDF file viewing program. Many commercial software packages can read netCDF files, such as ESRI ArcGIS and Matlab (from the Mathworks). A simple and free reader for OS X is Panoply[4], from NASA. If you want to really be able to interact with the outputs from this model, you should install:

- **Matplotlib/Pylab** (*http://matplotlib.sourceforge.net/*), a Matlab-like interactive plotting and analysis package, which uses Python as its "shell".

(section) 1.2 Building and Installing SatStress

RESUME HERE

[1] http://zaneselvans.org
[2] mailto:zane.selvans@colorado.edu
[3] http://www.unidata.ucar.edu
[4] http://www.giss.nasa.gov/tools/panoply/

Once you have the required software prerequisites installed, you should be able to `cd` into the directory containing the `SatStress` module, and simply type `make all` at the command line. This will compile the Love number code and run the small `SatStress.test` program embedded within `SatStress`, just to make sure that everything is in working order (or not). If you're not using the GNU Fortran 77 compiler `g77`, you'll need to edit the `Makefile` for the Love number code:

```
./Love/JohnWahr/Makefile
```

(section) 2 Design Overview

A few notes on the general architecture of the `SatStress` package.

(section) 2.1 A Toolkit, not a Program

The `SatStress` package is not itself a stand-alone program (or not much of one anyway). Instead it is a set of tools with which you can build programs that need to know about the stresses on the surface of a satellite, and how they compare to tectonic features, so you can do your own hypothesizing and testing.

(section) 2.2 Object Oriented

The package attempts to make use of object oriented programming[5] (OOP) in order to maximize the re-usability and extensibility of the code. Many scientists are more familiar with the imperative programming style[6] of languages like Fortran and C, but as more data analysis and hypothesis testing takes place inside computers, and as many scientists become highly specialized and knowledgeable software engineers (even if they don't want to admit it), the advantages of OOP become significant. If the object orientation of this module seems odd at first glance, don't despair, it's worth learning.

(section) 2.3 Written in Python

Python[7] is a general purpose, high-level scripting language. It is an interpreted language (as opposed to compiled languages like Fortran or C) and so Python code is very portable, meaning it is usable on a wide variety of computing platforms without any alteration. It is relatively easy to learn and easy to read, and it has a very active development community. It also has a large base of friendly, helpful scientific users and an enormous selection of pre-existing libraries designed for scientific applications. For those tasks which are particularly computationally intensive, Python allows you to extend the language with code written in C and Fortran. Python is also Free Software[8]. If you are a scientist and you write code, Python is a great choice.

(section) 2.4 Open Source

Because science today is intimately intertwined with computation, it is important for researchers to share the code that their scientific results are based on. No matter how elegant and accurate your derivation is, if your implementation of the model in code is wrong, your results will be flawed. As our models and hypotheses become more complex, our code becomes vital primary source material, and it needs to be open to peer review. Opening our source:

- allows bugs to be found and fixed more quickly
- facilitates collaboration and interoperability
- reduces duplicated effort
- enhances institutional memory

[5] http://en.wikipedia.org/wiki/Object-oriented_programming
[6] http://en.wikipedia.org/wiki/Imperative_programming
[7] http://www.python.org
[8] http://www.gnu.org/philosophy/free-sw.html

- encourages better software design and documentation

Of course, it also means that other people can use our code to write their own scientific papers, but *that is the fundamental nature of science*. We are all "standing on the shoulders of giants". Nobody re-derives quantum mechanics when they just want to do a little spectroscopy. Why should we all be re-writing each others code *ad nauseam*? Opening scientific source code will ultimately increase everyone's productivity. Additionally, a great deal of science is funded by the public, and our code is a major product of that funding. It is unethical to make it proprietary.

1.1 Modules

- **GridCalc**: Calculate stresses on an icy satellite over a rectangular geographic region on a regularly spaced lat-lon grid.
  (Section 2, p. 7)
- **SatStress**: A framework for calculating the surface stresses at a particular place and time on a satellite resulting from one or more tidal potentials.
  (Section 3, p. 9)
- **physcon**: A Python dictionary of physical constants in SI units.
  (Section 4, p. 59)

2 Module satstress.GridCalc

Calculate stresses on an icy satellite over a rectangular geographic region on a regularly spaced lat-lon grid.

The datacube containing the results of the calculation are output as a Unidata netCDF[9] (.nc) file, which can be displayed using a wide variety of visualization software.

2.1 Functions

main()
Calculate satellite stresses on a regular grid within a lat-lon window.

2.2 Class Grid

A container class defining the temporal and geographic range and resolution of the calculation.

The parameters defining the calculation grid are read in from a name value file, parsed into a Python dictionary using `SatStress.nvf2dict`, and used to set the data attributes of the `Grid` object.

The geographic extent of the calculation is specified by minimum and maximum values for latitude and longitude.

The geographic resolution of the calculation is defined by an angular separation between calculations. This angular separation is the same in the north-south and the east-west direction.

The temporal range and resolution of the calculation can be specified either in terms of actual time units (seconds) or in terms of the satellite's orbital position (in degrees). In both cases, time=0 is taken to occur at periapse.

Δ is a measure of how viscous or elastic the response of the body is. It's equal to $(\mu)/(\eta\omega)$ where μ and η are the shear modulus and viscosity of the surface layer, respectively, and ω is the forcing frequency to which the body is subjected (see Wahr et al. (2008) for a detailed discussion). It is a logarithmic parameter, so its bounds are specified as powers of 10, e.g. if the minimum value is -3, the initial Δ is 10ˆ-3 = 0.001.

2.2.1 Methods

__init__(*self*, *gridFile*, *satellite*=`None`)
Initialize the Grid object from a gridFile.

2.2.2 Instance Variables

Name	Description
grid_id	A string identifying the grid *(type=str)*

continued on next page

[9] http://www.unidata.ucar.edu/software/netcdf

Name	Description
lat_max	Northern bound, degrees (north positive). *(type=float)*
lat_min	Southern bound, degrees (north positive). *(type=float)*
latlon_step	Angular separation between calculations. *(type=float)*
lon_max	Eastern bound, degrees (east positive). *(type=float)*
lon_min	Western bound, degrees (east positive). *(type=float)*
nsr_delta_max	Final Δ = 10ˆ(nsr_delta_max) *(type=float)*
nsr_delta_min	Initial Δ = 10ˆ(nsr_delta_min) *(type=float)*
nsr_delta_numsteps	How many Δ values to calculate total *(type=int)*
orbit_max	Final orbital position in degrees (0 = periapse) *(type=float)*
orbit_min	Initial orbital position in degrees (0 = periapse) *(type=float)*
orbit_step	Orbital angular separation between calculations in degrees *(type=float)*
time_max	Final time at which calculation ends. *(type=float)*
time_min	Initial time at which calculation begins (0 = periapse). *(type=float)*
time_step	Seconds between subsequent calculations. *(type=float)*

3 Module satstress.SatStress

A framework for calculating the surface stresses at a particular place and time on a satellite resulting from one or more tidal potentials.

(section) 1 Input and Output

Because `SatStress` is a "library" module, it doesn't do a lot of input and output - it's mostly about doing calculations. It does need to read in the specification of a `Satellite` object though, and it can write the same kind of specification out. To do this, it uses name-value files, and a function called `nvf2dict`, which creates a Python dictionary (or "associative array").

A name-value file is just a file containing a bunch of name-value pairs, like:

```
ORBIT_ECCENTRICITY = 0.0094   # e must be < 0.25
```

It can also contain comments to enhance human readability (anything following a '#' on a line is ignored, as with the note in the line above).

(section) 2 Satellites

Obviously if we want to calculate the stresses on the surface of a satellite, we need to define the satellite, this is what the `Satellite` object does.

(section) 2.1 Specifying a Satellite

In order to specify a satellite, we need:

- an ID of some kind for the planet/satellite pair of interest
- the charactaristics of the satellite's orbital environment
- the satellite's internal structure and material properties
- the forcings to which the satellite is subjected

From a few basic inputs, we can calculate many derived characteristics, such as the satellite's orbital period or the surface gravity.

The internal structure and material properties are specified by a series of concentric spherical shells (layers), each one being homogeneous throughout its extent. Given the densities and thicknesses of the these layers, we can calculate the satellite's overall size, mass, density, etc.

Specifying a tidal forcing may be simple or complex. For instance, the `Diurnal` forcing depends only on the orbital eccentricity (and other orbital parameters already supplied), and the `NSR` forcing requires only the addition of the non-synchronous rotation period of the shell. Specifying an arbitrary true polar wander trajectory would be much more complex.

For the moment, becuase we are only including simple forcings, their specifying parameters are read in from the satellite definition file. If more, and more complex forcings are eventually added to the model, their specification will probably be split into a separate input file.

(section) 2.2 Internal Structure and Love Numbers

SatStress treats the solid portions of the satellite as viscoelastic Maxwell solids[10], that respond differently to forcings having different frequencies (ω). Given the a specification of the internal structure and material

[10] http://en.wikipedia.org/wiki/Maxwell_material

properties of a satellite as a series of layers, and information about the tidal forcings the body is subject to, it's possible to calculate appropriate Love numbers, which describe how the body responds to a change in the gravitational potential.

Currently the calculation of Love numbers is done by an external program written in Fortran by John Wahr and others, with roots reaching deep into the Dark Ages of computing. As that code (or another Love number code) is more closely integrated with the model, the internal structure of the satellite will become more flexible, but for the moment, we are limited to assuming a 4-layer structure:

- `ICE_UPPER`: The upper portion of the shell (cooler, stiffer)
- `ICE_LOWER`: The lower portion of the shell (warmer, softer)
- `OCEAN`: An inviscid fluid decoupling the shell from the core.
- `CORE`: The silicate interior of the body.

(section) 3 Stresses

`SatStress` can calculate the following stress fields:

1. `Diurnal`: stresses arising from an eccentric orbit, having a forcing frequency equal to the orbital frequency.
2. `NSR`: stresses arising due to the faster-than-synchronous rotation of a floating shell that is decoupled from the satellite's interior by a fluid layer (an ocean).

The expressions defining these stress fields are derived in "Modeling Stresses on Satellites due to Non-Synchronous Rotation and Orbital Eccentricity Using Gravitational Potential Theory" (preprint, 15MB PDF[11]) by Wahr et al. (submitted to *Icarus*, in March, 2008).

(section) 3.1 Stress Fields Live in `StressDef` Objects

Each of the above stress fields is defined by a similarly named `StressDef` object. These objects contain the formulae necessary to calculate the surface stress. The expressions for the stresses depend on many parameters which are defined within the `Satellite` object, and so to create a `StressDef` object, you need to provide a `Satellite` object.

There are many formulae which are identical for both the `NSR` and `Diurnal` stress fields, and so instead of duplicating them in both classes, they reside in the `StressDef` *base class*, from which all `StressDef` objects inherit many properties.

The main requirement for each `StressDef` object is that it must define the three components of the stress tensor τ:

- `Ttt` ($\tau\_\theta\theta$) the north-south (latitudinal) component
- `Tpt` ($\tau\_\phi\theta = \tau\_\theta\phi$) the shear component
- `Tpp` ($\tau\_\phi\phi$) the east-west (longitudinal) component

(section) 3.2 Stress Calculations are Performed by `StressCalc` Objects

Once you've *instantiated* a `StressDef` object, or several of them (one for each stress you want to include), you can compose them together into a `StressCalc` object, which will actually do calculations at given points on the surface, and given times, and return a 2x2 matrix containing the resulting stress tensor (each component of which is the sum of all of the corresponding components of the stress fields that were used to instantiated the `StressCalc` object).

[11] http://icymoons.com/Wahretal2008/stress.paper.pdf

This is (hopefully) easier than it sounds. With the following few lines, you can construct a satellite, do a single calculation on its surface, and see what it looks like:

```
>>> from SatStress import *
>>> the_sat = Satellite(open("input/Europa.satellite"))
>>> the_stresses = StressCalc([Diurnal(the_sat), NSR(the_sat)])
>>> Tau = the_stresses.tensor(theta=pi/4.0, phi=pi/3.0, t=10000)
>>> print(Tau)
```

The `SatStress.test` function shows a slightly more complex example, which should be enough to get you started using the package.

(section) 3.3 Extending the Model

Other stress fields can (and hopefully will!), be added easily, so long as they use the same mathematical definition of the membrane stress tensor (τ), as a function of co-latitude (θ) (measured south from the north pole), east-positive longitude (ϕ), measured from the meridian on the satellite which passes through the point on the satellite directly beneath the parent planet (assuming a synchronously rotating satellite), and time ($\boldsymbol{t}$), defined as seconds elapsed since pericenter.

This module could also potentially be extended to also calculate the surface strain (ϵ) and displacement ($\boldsymbol{s}$) fields, or to calculate the stresses at any point within the satellite.

Date: Fri Mar 28 20:14:34 2008

Author: Zane Selvans

Contact: zane.selvans@colorado.edu

3.1 Functions

nvf2dict(*nvf*, *comment*=`'#'`)

Reads from a file object listing name value pairs, creating and returning a corresponding Python dictionary.

The file should contain a series of name value pairs, one per line separated by the '=' character, with names on the left and values on the right. Blank lines are ignored, as are lines beginning with the comment character (assumed to be the pound or hash character '#', unless otherwise specified). End of line comments are also allowed. String values should not be quoted in the file. Names are case sensitive.

Returns a Python dictionary that uses the names as keys and the values as values, and so all Python limitations on what can be used as a dictionary key apply to the name fields.

Leading and trailing whitespace is stripped from all names and values, and all values are returned as strings.

Parameters

`nvf`: an open file object from which to read the name value pairs

(type=file)

`comment`: character which begins comments

(type=str)

Return Value

a dictionary containing the name value pairs read in from `nvf`.

(type=dict)

Raises

`NameValueFileParseError` if a non-comment input line does not contain an '=' character, or if a non-comment line has nothing but whitespace preceeding or following the '=' character.

`NameValueFileDuplicateNameError` if more than one instance of the same name is found in the input file `nvf`.

test(*argv*=['(imported)'])

Check to see that SatStress gives the expected output from a series of known calculations.

Calculates the stresses due to the `NSR` and `Diurnal` forcings at a series of lat lon points on Europa, over the course of most of an orbit, and also at a variety of different amounts of viscous relaxation. Compares the calculated values to those listed in the `pickle` file passed in via the command line.

`test` is called from the `SatStress Makefile`, when one does `make test`, with the appropriate `pickle`d input to compare against (it is provided with the source code).

This function also acts as a short demonstration of how to use the SatStress module.

Parameters

`argv:` a list of command line arguments

(type=list)

Return Value

`True` if the test fails. `False` if the test passes.

(type=bool)

3.2 Class Satellite

object
└ **satstress.SatStress.Satellite**

An object describing the physical structure and context of a satellite.

Defines a satellite's material properties, internal structure, orbital context, and the tidal forcings to which it is subjected.

3.2.1 Methods

__init__(*self*, *satFile*)

Construct a Satellite object from a satFile

(section) Required input file parameters:

The Satellite is initialized from a name value file (as described under `nvf2dict`). The file must define the following parameters, all of which are specified in SI (MKS) units.

- `SYSTEM_ID`: A string identifying the planetary system, e.g. JupiterEuropa.
- `PLANET_MASS`: The mass of the planet the satellite orbits [kg].
- `ORBIT_ECCENTRICITY`: The eccentricity of the satellite's orbit. Must not exceed 0.25.
- `ORBIT_SEMIMAJOR_AXIS`: The semimajor axis of the satellite's orbit [m].
- `NSR_PERIOD`: The time it takes for the satellite's icy shell to undergo one full rotation [s]. If you don't want to have any NSR stresses, just put INFINITY here.
- `LOVE_PATH`: The path to the program used to calculate the frequency-dependent degree-2 Love numbers. Currently this code is one provided by John Wahr.

Parameters

`satFile`: Open file object containing name value pairs specifying the satellite's internal structure and orbital context, and the tidal forcings to which the satellite is subjected.

(type=file)

Return Value

a Satellite object corresponding to the proffered input file.

(type=`Satellite`)

Raises

`NameValueFileError` if parsing of the input file fails.

`MissingSatelliteParamError` if a required input field is not found within the input file.

`NonNumberSatelliteParamError` if a required input which is of a numeric type is found within the file, but its value is not convertable to a float.

`LoveLayerNumberError` if the number of layers specified is not exactly 4.

`GravitationallyUnstableSatelliteError` if the layers are found not to decrease in density from the core to the surface.

`ExcessiveSatelliteMassError` if the mass of the satellite's parent planet is not at least 10 times larger than the mass of the satellite.

`LargeEccentricityError` if the satellite's orbital eccentricity is greater than 0.25

`NegativeNSRPeriodError` if the NSR period of the satellite is less than zero.

`IOError` if the file specified in `LOVE_PATH` is not openable.

Overrides: object.__init__

mass(*self*)

Calculate the mass of the satellite. (the sum of the layer masses)

radius(*self*)

Calculate the radius of the satellite (the sum of the layer thicknesses).

density(*self*)

Calculate the mean density of the satellite in [kg m^-3].

surface_gravity(*self*)

Calculate the satellite's surface gravitational acceleration in [m s^-2].

orbit_period(*self*)

Calculate the satellite's Keplerian orbital period in seconds.

mean_motion(*self*)

Calculate the orbital mean motion of the satellite [rad s^-1].

__str__(*self*)

Output a satellite definition file equivalent to the object.

Overrides: object.__str__

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__()

3.2.2 Properties

Name	Description
Inherited from object __class__	

3.2.3 Instance Variables

Name	Description
layers	a list of `SatLayer` objects, describing the layers making up the satellite. The layers are ordered from the center of the satellite outward, with layers[0] corresponding to the core. *(type=list)*

continued on next page

Name	Description
love_path	the path to the program which will be used to calculate the degree-2, complex, frequency dependent Love numbers h2, and k2. Path is relative to the directory in which the program is running. Corresponds to `LOVE_PATH` in the input file. *(type=str)*
nsr_period	the time it takes for the decoupled ice shell to complete one full rotation [s], corresponds to `NSR_PERIOD` in the input file. May also be set to infinity (inf, infinity, INF, INFINITY). *(type=float)*
num_layers	the number of layers making up the satellite, as indicated by the number of keys within the satParams dictionary contain the string 'LAYER_ID'. Currently this must equal 4. *(type=int)*
orbit_eccentricity	the satellite's orbital eccentricity, corresponds to `ORBIT_ECCENTRICITY` in the input file. *(type=float)*
orbit_semimajor_axis	semimajor axis of the satellite's orbit [m], corresponds to `ORBIT_SEMIMAJOR_AXIS` in the input file. *(type=float)*
planet_mass	the mass of the planet the satellite orbits [kg], corresponds to `PLANET_MASS` in the input file. *(type=float)*
satParams	dictionary containing the name value pairs read in from the input file. *(type=dict)*
sourcefile	the file object which was read in order to create the `Satellite` instance. *(type=file)*
system_id	string identifying the planet/satellite system, corresponds to `SYSTEM_ID` in the input file. *(type=str)*

3.3 Class SatLayer

object
        satstress.SatStress.SatLayer

An object describing a uniform material layer within a satellite.

Note that a layer by itself has no knowledge of where within the satellite it resides. That information is contained in the ordering of the list of layers within the satellite object.

3.3.1 Methods

__init__(*self*, *sat*, *layer_n*=0)

Construct an object representing a layer within a `Satellite`.

Gets values from the `Satellite.satParams` dictionary for the layer that corresponds to the value of `layer_n`.

Each layer is defined by seven parameter values, and each layer has a unique numeric identifier, appended to the end of all the names of its parameters. Layer zero is the core, with the number increasing as the satellite is built up toward the surface. In the below list the "N" at the end of the parameter names should be replaced with the number of the layer (`layer_n`). Currently, because of the constraints of the Love number code that we are using, you must specify 4 layers (`CORE`, `OCEAN`, `ICE_LOWER`, `ICE_UPPER`).

- `LAYER_ID_N`: A string identifying the layer, e.g. `OCEAN` or `ICE_LOWER`
- `DENSITY_N`: The density of the layer at zero pressure [m kg^-3]
- `LAME_MU_N`: The real-valued Lame parameter μ (shear modulus) [Pa]
- `LAME_LAMBDA_N`: The real-valued Lame parameter λ [Pa]
- `THICKNESS_N`: The thickness of the layer [m]
- `VISCOSITY_N`: The viscosity of the layer [Pa s]
- `TENSILE_STRENGTH_N`: The tensile strength of the layer [Pa]

Not all layers necessarily require all parameters in order for the calculation to succeed, but it is required that they be provided. Parameters that will currently be ignored:

- `TENSILE_STRENGTH` of all layers except for the surface (which is only used when creating fractures).
- `VISCOSITY` of the ocean and the core.
- `LAME_MU` of the ocean, assumed to be zero.

Parameters

`sat`: the `Satellite` object to which the layer belongs.

(type=`Satellite`)

`layer_n`: layer_n indicates which layer in the satellite is being defined, with n=0 indicating the center of the satellite, and increasing outward. This is needed in order to select the appropriate values from the `Satellite.satParams` dictionary.

(type=int)

Return Value

a `SatLayer` object having the properties specified in the

(type=`SatLayer`)

Raises

`MissingSatelliteParamError` if any of the seven input parameters listed above is not found in the `Satellite.satParams`

__str__(*self*)

Output a human and machine readable text description of the layer.

Note that because the layer object does not know explicitly where it is within the stratified Satellite (that information is contained in the ordering of Satellite.layers list), this method cannot be used in the output of a Satellite object.

Overrides: object.__str__

maxwell_time(*self*)

Calculate the Maxwell relaxation time of the layer [s] (viscosity/lame_mu).

bulk_modulus(*self*)

Calculate the bulk modulus (κ) of the layer [Pa].

youngs_modulus(*self*)

Calculate the Young's modulus (E) of the layer [Pa].

poissons_ratio(*self*)

Calculate poisson's ratio (ν) of the layer [Pa].

p_wave_velocity(*self*)

Calculate the velocity of a compression wave in the layer [m s^-1]

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__()

3.3.2 Properties

Name	Description
Inherited from object	
__class__	

3.3.3 Instance Variables

Name	Description
density	the density of the layer, at zero pressure [kg m^-3], *(type=float)*
lame_lambda	the layer's Lame parameter, λ [Pa]. *(type=float)*
lame_mu	the layer's Lame parameter, μ (the shear modulus) [Pa]. *(type=float)*
layer_id	a string identifying the layer, e.g. `CORE`, or `ICE_LOWER` *(type=str)*
tensile_str	the tensile failure strength of the layer [Pa]. *(type=float)*
thickness	the radial thickness of the layer [m]. *(type=float)*
viscosity	the viscosity of the layer [Pa s]. *(type=float)*

3.4 Class LoveNum

object ─┐
satstress.SatStress.LoveNum

A container class for the complex Love numbers: h2, k2, and l2.

3.4.1 Methods

__init__(*self*, *h2_real*, *h2_imag*, *k2_real*, *k2_imag*, *l2_real*, *l2_imag*)

Using the real and imaginary parts, create complex values.

Overrides: object.__init__

__str__(*self*)

Return a human readable string representation of the Love numbers

Overrides: object.__str__

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__()

3.4.2 Properties

Name	Description
Inherited from object __class__	

3.4.3 Instance Variables

Name	Description
h2	the degree 2 complex, frequency dependent Love number h. *(type=complex)*
k2	the degree 2 complex, frequency dependent Love number k. *(type=complex)*
l2	the degree 2 complex, frequency dependent Love number l. *(type=complex)*

3.5 Class StressDef

object ─┐
satstress.SatStress.StressDef

Known Subclasses: satstress.SatStress.Diurnal, satstress.SatStress.NSR

A base class from which particular tidal stress field objects descend.

Different tidal forcings are specified as sub-classes of this superclass (one for each separate forcing).

In the expressions of the stress fields, the time t is specified in seconds, with zero occuring at periapse, in order to be compatible with the future inclusion of stressing mechanisms which may have explicit time dependence instead of being a function of the satellite's orbital position (e.g. a true polar wander trajectory).

Location is specified within a polar coordinate system having its origin at the satellite's center of mass, using the following variables:

- co-latitude (θ): The arc separating a point on the surface of the satellite from the north pole ($0 < \theta < \pi$).
- longitude (ϕ): The arc separating the meridian of a point and the meridian which passes under the average location of the primary (planet) in the sky over the course of an orbit

$(0 < \phi < 2\pi)$. **East is taken as positive.**

Each subclass must define its own version of the three components of the membrane stress tensor, `Ttt`, `Tpp`, and `Tpt` (the north-south, east-west, and shear stress components) as methods.

3.5.1 Methods

calcLove(*self*)

Calculate the Love numbers for the satellite and the given forcing.

If an infinite forcing period is given, return zero valued Love numbers.

This is a wrapper function, which can be used to call different Love number codes in the future.

Raises

`InvalidLoveNumberError` if the magnitude of the imaginary part of any Love number is larger than its real part, if the real part is ever less than zero, or if the real coefficient of the imaginary part is ever positive.

calcLoveInfinitePeriod(*self*)

Return a set of zero Love numbers constructed statically.

This method is included so we don't have to worry about whether the Love number code can deal with being given an infinite period. All stresses will relax to zero with an infinite period (since the shear modulus μ goes to zero), so it doesn't really matter what we set the Love numbers to here.

calcLoveWahr4LayerExternal(*self*)

Use John Wahr's Love number code to calculate h, k, and l.

This is done by an external program, written in Fortran by John Wahr (and others), and called elsewhere on the system. The path to this external program is specified in the satellite definition file as the parameter `LOVE_PATH`. At the moment, the code is fairly limited in the kind of input it can take. The specified satellite must:

- use a Maxwell rheology
- have a liquid water ocean underlying the ice shell
- have a 4-layer structure (ice_upper, ice_lower, ocean, core)

Eventually the Love number code will be more closely integrated with this package, allowing more flexibility in the interior structure of the satellite.

A temporary directory named lovetmp-XXXXXXX (where the X's are a random hexadecimal number) is created in the current working directory, within which the Love number code is run. The directory is deleted immediately following the calculation.

Raises

`LoveExcessiveDeltaError` if `StressDef.Delta`() > 10^9 for either of the ice layers.

Delta(*self*, *layer_n*=`-1`)

Calculate Δ, a measure of how viscous the layer's response is.

Parameters

`layer_n`: indicates which satellite layer Delta should be calculated for, defaulting to the surface (recall that layer 0 is the core)

(type=int)

Return Value

$\Delta = \mu/(\omega * \eta)$

(type=float)

Z(*self*)

Calculate the value of Z, a constant that sits in front of many terms in the potential defined by Wahr et al. (2008).

Return Value

Z, a common constant in many of the Wahr et al. potential terms.

(type=float)

mu_twiddle(*self*, *layer_n*=`-1`)

Calculate the frequency-dependent Lame parameter μ for a Maxwell rheology.

Parameters

`layer_n`: number of layer for which we want to calculate μ, defaults to the surface (with the core being layer zero).

Return Value

the frequency-dependent Lame parameter μ for a Maxwell rheology

(type=complex)

lambda_twiddle(*self*, *layer_n*=`-1`)

Calculate the frequency-dependent Lame parameter λ for a Maxwell rheology.

Parameters

`layer_n`: number of layer for which we want to calculate μ, defaults to the surface (with the core being layer zero).

Return Value

the frequency-dependent Lame parameter λ for a Maxwell rheology.

(type=complex)

alpha(*self*)

Calculate the coefficient alpha twiddle for the surface layer (see Wahr et al. 2008).

Return Value

Calculate the coefficient alpha twiddle for the surface layer (see Wahr et al. 2008).

(type=complex)

Gamma(*self*)

Calculate the coefficient capital Gamma twiddle for the surface layer (see Wahr et al. 2008).

Return Value

the coefficient capital Gamma twiddle for the surface layer (see Wahr et al. 2008).

(type=complex)

b1(*self*)

Calculate the coefficient beta one twiddle for the surface layer (see Wahr et al. 2008).

Return Value

the coefficient beta one twiddle for the surface layer (see Wahr et al. 2008).

(type=complex)

g1(*self*)

Calculate the coefficient gamma one twiddle for the surface layer (see Wahr et al. (2008)).

Return Value

the coefficient gamma one twiddle for the surface layer (see Wahr et al. (2008)).

(type=complex)

b2(*self*)

Calculate the coefficient beta two twiddle for the surface layer (see Wahr et al. (2008)).

Return Value

the coefficient beta two twiddle for the surface layer (see Wahr et al. (2008)).

(type=complex)

g2(*self*)

Calculate the coefficient gamma two twiddle for the surface layer (see Wahr et al. (2008)).

Return Value

the coefficient gamma two twiddle for the surface layer (see Wahr et al. (2008)).

(type=complex)

Ttt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\theta\theta$ (north-south) component of the stress tensor.

In the base class, this is a purely virtual method - it must be defined by the subclasses that describe particular tidal stresses.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

(type=float)

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

(type=float)

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

(type=float)

Return Value

the $\tau\_\theta\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Tpp(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\phi$ (east-west) component of the stress tensor.

In the base class, this is a purely virtual method - it must be defined by the subclasses that describe particular tidal stresses.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

(type=float)

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

(type=float)

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

(type=float)

Return Value

the $\tau\_\phi\phi$ component of the 2x2 membrane stress tensor.

(type=float)

Tpt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\theta$ (off-diagonal) component of the stress tensor.

In the base class, this is a purely virtual method - it must be defined by the subclasses that describe particular tidal stresses.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

(type=float)

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

(type=float)

`t`: the time in seconds elapsed since pericenter, at which to perform the stress calculation [s].

(type=float)

Return Value

the $\tau\_\phi\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __str__()

3.5.2 Properties

Name	Description
Inherited from object __class__	

3.5.3 Class Variables

Name	Description
omega	the forcing frequency associated with the stress. **Value:** `0.0` *(type=float)*
satellite	the satellite which the stress is being applied to. **Value:** `None` *(type=`Satellite`)*
love	the Love numbers which result from the given forcing frequency and the specified satellite structure. **Value:** `LoveNum(0, 0, 0, 0, 0, 0)` *(type=`LoveNum`)*

3.6 Class NSR

object ┐

satstress.SatStress.StressDef ┐

satstress.SatStress.NSR

An object defining the stress field which arises from the non-synchronous rotation (NSR) of a satellite's icy shell.

NSR is a subclass of `StressDef`. See the derivation and detailed discussion of this stress field in in Wahr et al. (2008).

3.6.1 Methods

__init__(*self*, *satellite*)

Initialize the definition of the stresses due to NSR of the ice shell.

The forcing frequency ω is the frequency with which a point on the surface passes through a single hemisphere, because the NSR stress field is degree 2 (that is, it's 2x the expected ω from a full rotation)

Because the core is not subject to the NSR forcing (it remains tidally locked and synchronously rotating), all stresses within it are presumed to relax away, allowing it to deform into a tri-axial ellipsoid, with its long axis pointing toward the parent planet. In order to allow for this relaxation the shear modulus (μ) of the core is set to an artificially low value for the purpose of the Love number calculation. This increases the magnitude of the radial deformation (and the Love number h2) significantly. See Wahr et al. (2008) for complete discussion.

Parameters

`satellite`: the satellite to which the stress is being applied.

(type=`Satellite`)

Return Value

an object defining the NSR stresses for a particular satellite.

(type=`NSR`)

Overrides: object.__init__

Ttt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\theta\theta$ (north-south) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\theta\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Ttt

Tpp(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\phi$ (east-west) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\phi\phi$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Tpp

Tpt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\theta$ (off-diagonal) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\phi\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Tpt

Inherited from satstress.SatStress.StressDef(Section 3.5)

Delta(), Gamma(), Z(), alpha(), b1(), b2(), calcLove(), calcLoveInfinitePeriod(), calcLoveWahr4LayerExternal(), g1(), g2(), lambda_twiddle(), mu_twiddle()

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __str__()

3.6.2 Properties

Name	Description
Inherited from object	
__class__	

3.6.3 Class Variables

Name	Description
Inherited from satstress.SatStress.StressDef (Section 3.5)	
love, omega, satellite	

3.7 Class Diurnal

object
satstress.SatStress.StressDef
satstress.SatStress.Diurnal

An object defining the stress field that arises on a satellite due to an eccentric orbit.

Diurnal is a subclass of `StressDef`. See the derivation and detailed discussion of this stress field in in Wahr et al. (2008).

3.7.1 Methods

__init__(*self*, *satellite*)

Sets the object's satellite and omega attributes; calculates Love numbers.

Parameters

`satellite`: the satellite to which the stress is being applied.

(type=`Satellite`)

Return Value

an object defining the NSR stresses for a particular satellite.

(type=`NSR`)

Overrides: object.__init__

Ttt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\theta\theta$ (north-south) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\theta\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Ttt

Tpp(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\phi$ (east-west) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time, in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\phi\phi$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Tpp

Tpt(*self*, *theta*, *phi*, *t*)

Calculates the $\tau\_\phi\theta$ (off-diagonal) component of the stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

`t`: the time in seconds elapsed since pericenter, at which to perform the stress calculation [s].

Return Value

the $\tau\_\phi\theta$ component of the 2x2 membrane stress tensor.

(type=float)

Overrides: satstress.SatStress.StressDef.Tpt

Inherited from satstress.SatStress.StressDef(Section 3.5)

Delta(), Gamma(), Z(), alpha(), b1(), b2(), calcLove(), calcLoveInfinitePeriod(), calcLoveWahr4LayerExternal(), g1(), g2(), lambda_twiddle(), mu_twiddle()

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __str__()

3.7.2 Properties

Name	Description
Inherited from object	
__class__	

3.7.3 Class Variables

Name	Description
Inherited from satstress.SatStress.StressDef (Section 3.5)	
love, omega, satellite	

3.8 Class StressCalc

object ─┐
satstress.SatStress.StressCalc

An object which calculates the stresses on the surface of a `Satellite` that result from one or more stress fields.

3.8.1 Methods

__init__(*self*, *stressdefs*)

Defines the list of stresses which are to be calculated at a given point.

Parameters

`stressdefs`: a list of `StressDef` objects, corresponding to the stresses which are to be included in the calculation.

(type=list)

Return Value

(type=`StressCalc`)

Overrides: object.__init__

tensor(*self*, *theta*, *phi*, *t*)

Calculates surface stresses and returns them as a 2x2 stress tensor.

Parameters

`theta`: the co-latitude of the point at which to calculate the stress [rad].

(type=float)

`phi`: the east-positive longitude of the point at which to calculate the stress [rad].

(type=float)

`t`: the time in seconds elapsed since pericenter, at which to perform the stress calculation [s].

(type=float)

Return Value

symmetric 2x2 surface (membrane) stress tensor τ

(type=Numpy.array)

Inherited from object

__delattr__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __str__()

3.8.2 Properties

Name	Description
Inherited from object __class__	

3.8.3 Instance Variables

Name	Description
stresses	a list of `StressDef` objects, corresponding to the stresses which are to be included in the calculations done by the `StressCalc` object. *(type=list)*

3.9 Class Error

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error

Known Subclasses: satstress.SatStress.SatelliteParamError, satstress.SatStress.NameValueFileError

Base class for errors within the SatStress module.

3.9.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

Inherited from object

__hash__(), __reduce_ex__()

3.9.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.10 Class NameValueFileError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.NameValueFileError

Known Subclasses: satstress.SatStress.NameValueFileDuplicateNameError, satstress.SatStress.NameVa

Base class for errors related to NAME=VALUE style input files.

3.10.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

Inherited from object

__hash__(), __reduce_ex__()

3.10.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.11 Class NameValueFileParseError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.NameValueFileError
satstress.SatStress.NameValueFileParseError

Indicates a poorly formatted NAME=VALUE files.

3.11.1 Methods

__init__(*self*, *nvf*, *line*)

Stores the file and line that generated the parse error.

The file object (nvf) and the contents of the poorly formed line (badline) are stored within the exception, so we can print an error message with useful debugging information to the user.

Overrides: object.__init__

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.11.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.12 Class NameValueFileDuplicateNameError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.NameValueFileError
satstress.SatStress.NameValueFileDuplicateNameErro

Indicates multiple copies of the same name in an input file.

3.12.1 Methods

__init__(*self*, *nvf*, *name*)

Stores the file and the NAME that was found to be multiply defined.

NAME is the key, which has been found to be multiply defined in the input file, nvf.

Overrides: object.__init__

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.12.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.13 Class SatelliteParamError

object

exceptions.BaseException

exceptions.Exception

satstress.SatStress.Error

satstress.SatStress.SatelliteParamError

Known Subclasses: satstress.SatStress.InvalidSatelliteParamError, satstress.SatStress.MissingSatelliteP

Indicates a problem with the Satellite initialization.

3.13.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

Inherited from object

__hash__(), __reduce_ex__()

3.13.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.14 Class MissingSatelliteParamError

object

exceptions.BaseException

exceptions.Exception

satstress.SatStress.Error

satstress.SatStress.SatelliteParamError

satstress.SatStress.MissingSatelliteParamError

Indicates a required parameter was not found in the input file.

3.14.1 Methods

__init__(*self*, *sat*, *missingname*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.14.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.15 Class InvalidSatelliteParamError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError

Known Subclasses: satstress.SatStress.ExcessiveSatelliteMassError, satstress.SatStress.GravitationallyU satstress.SatStress.InvalidLoveNumberError, satstress.SatStress.LargeEccentricityError, satstress.SatStress.LoveExcessiveDeltaError, satstress.SatStress.LoveLayerNumberError, satstress.SatStress.I satstress.SatStress.LowLayerThicknessError, satstress.SatStress.NegativeLayerParamError, satstress.SatStress.NegativeNSRPeriodError, satstress.SatStress.NonNumberSatelliteParamError

Raised when a required parameter is not found in the input file.

3.15.1 Methods

__init__(*self*, *sat*)
Default initialization of an InvalidSatelliteParamError Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object. Overrides: object.__init__

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

Inherited from object

__hash__(), __reduce_ex__()

3.15.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.16 Class LargeEccentricityError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.LargeEccentricityError

Raised when satellite orbital eccentricity is > 0.25

3.16.1 Methods

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from satstress.SatStress.InvalidSatelliteParamError(Section 3.15)

__init__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.16.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.17 Class NegativeNSRPeriodError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.NegativeNSRPeriodError

Raised if the satellite's NSR period is less than zero

3.17.1 Methods

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from satstress.SatStress.InvalidSatelliteParamError(Section 3.15)

__init__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.17.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.18 Class ExcessiveSatelliteMassError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.ExcessiveSatelliteMassError

Raised if the satellite's parent planet is less than 10x as massive as the satellite.

3.18.1 Methods

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from satstress.SatStress.InvalidSatelliteParamError(Section 3.15)

__init__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.18.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.19 Class LoveLayerNumberError

object

exceptions.BaseException

exceptions.Exception

satstress.SatStress.Error

satstress.SatStress.SatelliteParamError

satstress.SatStress.InvalidSatelliteParamError

satstress.SatStress.LoveLayerNumberError

Raised if the number of layers specified in the satellite definition file is incompatible with the Love number code.

3.19.1 Methods

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from satstress.SatStress.InvalidSatelliteParamError(Section 3.15)

__init__()

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.19.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.20 Class InvalidLoveNumberError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.InvalidLoveNumberError

Raised if the Love numbers are found to be suspicious.

3.20.1 Methods

__init__(*self*, *stress*, *love*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.20.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.21 Class LoveExcessiveDeltaError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.LoveExcessiveDeltaError

Raised when $\Delta > 10\hat{}9$ for any of the ice layers, at which point the Love number code becomes unreliable.

3.21.1 Methods

__init__(*self*, *stress*, *layer_n*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.21.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.22 Class GravitationallyUnstableSatelliteError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.GravitationallyUnstableSatelli

Raised if the density of layers is found not to decrease as you move toward the surface from the center of the satellite.

3.22.1 Methods

__init__(*self*, *sat*, *layer_n*)

Overrides the base InvalidSatelliteParamError.__init__() function.

We also need to know which layers have a gravitationally unstable arrangement.

Overrides: object.__init__

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.22.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.23 Class NonNumberSatelliteParamError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.NonNumberSatelliteParamErr

Indicates that a non-numeric value was found for a numerical parameter.

3.23.1 Methods

__init__(*self*, *sat*, *badname*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.23.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.24 Class LowLayerDensityError

object ─┐
exceptions.BaseException ─┐
exceptions.Exception ─┐
satstress.SatStress.Error ─┐
satstress.SatStress.SatelliteParamError ─┐
satstress.SatStress.InvalidSatelliteParamError ─┐
satstress.SatStress.LowLayerDensityError

Indicates that a layer has been assigned an unrealistically low density.

3.24.1 Methods

__init__(*self*, *sat*, *layer_n*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.24.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

3.25 Class LowLayerThicknessError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.LowLayerThicknessError

Indicates that a layer has been given too small of a thickness

3.25.1 Methods

__init__(*self*, *sat*, *layer_n*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.25.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

3.26 Class NegativeLayerParamError

object
exceptions.BaseException
exceptions.Exception
satstress.SatStress.Error
satstress.SatStress.SatelliteParamError
satstress.SatStress.InvalidSatelliteParamError
satstress.SatStress.NegativeLayerParamError

Indicates a layer has been given an unphysical material property.

3.26.1 Methods

__init__(*self*, *sat*, *badparam*)

Default initialization of an InvalidSatelliteParamError

Simply sets self.sat = sat (a Satellite object). Most errors can be well described using only the parameters stored in the Satellite object.

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__()

Inherited from object

__hash__(), __reduce_ex__()

3.26.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

4 Module satstress.physcon

A Python dictionary of physical constants in SI units.

Values are from CODATA 2006[12]

Each item in the dictionary consists of a string key, and a list value, containing the following:

- description (string)
- symbol (string)
- value (float)
- sd (float)
- relative sd (float)
- value(sd) unit (string)
- source (string)

Author: Herman J.C. Berendsen

Contact: herman@hjcb.nl

Copyright: Copyright 2004, 2007 Herman J.C. Berendsen

License: GNU General Public License version 3

4.1 Functions

descr(*key*)

help()

relsd(*key*)

[12] http://www.physics.nist.gov/cuu/Constants/
[13] http://www.hjcb.nl/python
[14] mailto:herman@hjcb.nl

sd(*key*)

value(*key*)

4.2 Variables

Name	Description
F	**Value:** `96485.3399`
G	**Value:** `6.67428e-11`
N_A	**Value:** `6.02214179e+23`
R	**Value:** `8.314472`
a_0	**Value:** `5.2917720859e-11`
all	**Value:** `{'alpha': ['fine-structure constant = e^2/(4 pi eps_0 hba...`
alpha	**Value:** `0.0072973525376`
c	**Value:** `299792458.0`
cloc	**Value:** `299792458.0`
e	**Value:** `1.60217653e-19`
eps_0	**Value:** `8.85418781762e-12`
g_e	**Value:** `-2.00231930436`
g_p	**Value:** `5.585694713`
gamma_p	**Value:** `267522209.9`
h	**Value:** `6.62606896e-34`
hbar	**Value:** `1.054571628e-34`
k_B	**Value:** `1.3806504e-23`
m_d	**Value:** `3.3435832e-27`
m_e	**Value:** `9.10938215e-31`
m_n	**Value:** `1.674927211e-27`
m_p	**Value:** `1.674927211e-27`
mu_0	**Value:** `1.25663706144e-06`
mu_B	**Value:** `9.27400915e-24`
mu_N	**Value:** `5.05078324e-27`
mu_e	**Value:** `-9.28476377e-24`
mu_p	**Value:** `1.41060662e-26`
pi	**Value:** `3.14159265359`
sigma	**Value:** `5.6704e-08`
u	**Value:** `1.660538782e-27`

Index