stash Documentation

Release 0.0.dev036

Brett Morris

Jul 20, 2018

Contents

This is the documentation for `stash`, which Simulates Transits Acquired by SDO HMI in continuum imaging mode. This is a collaboration between Brett Morris, Monica Bobra, and Manodeep Sinha.

Warning: Stash is currently in active development. It may break at any time!

Part I

stash Documentation

This is the documentation for stash.

CHAPTER 1

Reference/API

1.1 stash Package

1.1.1 Functions

simulate_lightcurve(image, period, a, b[, ...]) Simulate a light curve of a planet with radius R_planet_physical with orbital period period, semimajor axis a, and assuming the Sun has radius R_star_physical.

test(**kwargs) Run the tests for the package.

transit_duration(R_star, R_planet, ...) Compute transit duration from first through fourth contact given orbital and physical parameters.

generate_lightcurve

stash.**generate_lightcurve**()

Parameters

image
[ndarray] SDO HMI image

R_planet_pixels
[float] Radius of planet in pixels

background
[float] Backgorund flux in counts

supersample_factor
[int] Supersample the planet pixelation by a factor `supersample_factor`

Returns

fluxes
[`ndarray`] Array of fluxes

simulate_lightcurve

`stash.`**`simulate_lightcurve`**(*image*, *period*, *a*, *b*, *R_planet_physical=<<class 'astropy.constants.iau2015.IAU2015'> name='Nominal Earth equatorial radius' value=6378100.0 uncertainty=0.0 unit='m' reference='IAU 2015 Resolution B 3'>*, *background=269*, *R_star_physical=<<class 'astropy.constants.iau2015.IAU2015'> name='Nominal solar radius' value=695700000.0 uncertainty=0.0 unit='m' reference='IAU 2015 Resolution B 3'>*, *supersample_factor=1*, *sdo_hmi=True*)

Simulate a light curve of a planet with radius `R_planet_physical` with orbital period `period`, semimajor axis `a`, and assuming the Sun has radius `R_star_physical`.

Parameters

image
[`ndarray`] SDO HMI continuum image of the Sun

period
[`Quantity`] Orbital period of the planet

a
[`Quantity`] Semimajor axis of the planet in absolute units

b
[float] Impact parameter (defined on [0, 1])

R_planet_physical
[`Quantity`] Radius of the planet

R_star_physical
[`Quantity`] Radius of the star

Returns

lc
[`LightCurve`] Simulated light curve of a planet transiting the star in `image`.

test

`stash.`**`test`**(***kwargs*)

Run the tests for the package.

This method builds arguments for and then calls `pytest.main`.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. Accepts comma separated string to specify multiple packages. If nothing is specified all default tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int, optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

pastebin
[('failed', 'all', None), optional] Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @pytest.mark.remote_data. This can be set to run no tests with remote data (`none`), only ones that use data from http://data.astropy.org (`astropy`), or all tests that use remote data (`any`). The default is `none`.

repeat
[`int`, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[`bool`, optional] When `True`, skips running the doctests in the .rst files.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

transit_duration

stash.**transit_duration**(*R_star*, *R_planet*, *orbital_period*, *semimajor_axis*, *impact_parameter*)
Compute transit duration from first through fourth contact given orbital and physical parameters.

Parameters

R_star
[Quantity] Stellar radius

R_planet
[Quantity] Planet radius

orbital_period
[Quantity] Orbital period

semimajor_axis
[Quantity] Orbital semimajor axis

impact_parameter
[float] Impact parameter (-1, 1)

Returns

duration
[Quantity] Transit duration from first through fourth contact.

1.1.2 Classes

LightCurve(times, fluxes)	Container for light curves.
UnsupportedPythonError	

LightCurve

class stash.**LightCurve**(*times*, *fluxes*)
Bases: object

Container for light curves.

Parameters

times
[ndarray] Times

fluxes
[ndarray] Fluxes

Methods Summary

get_transit_model(init_params, yerr)	Fit a Mandel & Agol transit model to the light curve.
plot(*args, **kwargs)	

Methods Documentation

get_transit_model(*init_params*, *yerr*)
Fit a Mandel & Agol transit model to the light curve.

Free parameters: inclination, midtransit time, limb-darkening parameters

Parameters

init_params
[list of length 4] Initial fitting parameters for inclination, midtransit time, and two quadratic limb-darkening parmaeters

yerr
[float] Uncertainty on flux measurements

Returns

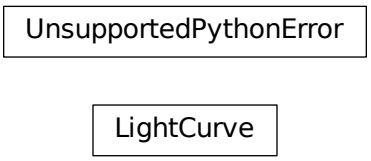
lc
[LightCurve] Best-fit transit model

plot(**args*, ***kwargs*)

UnsupportedPythonError

exception stash.**UnsupportedPythonError**

1.1.3 Class Inheritance Diagram



Part II

Installing stash

CHAPTER 2

The easy way

You can install stash via pip like so:

```
pip install git+git://github.com/bmorris3/stash.git
```

CHAPTER 3

Manually building stash

Stash requires some python packages to be installed first, which you can get via pip like so:

```
pip install numpy batman-package matplotlib sunpy astropy
```

Once you have those installed, you can install the latest version of `stash` by building from the source:

```
git clone https://github.com/bmorris3/stash.git
cd stash
python setup.py install
```

Part III

Getting Started

CHAPTER 4

Fetching an image with Fido

For this demo, we'll be using sunpy and matplotlib. Let's do some imports:

```
import matplotlib.pyplot as plt
from sunpy.net import Fido, attrs as a
from sunpy.map import Map
```

Great! Now we'll use Fido to search between two times for an image taken by SDO/HMI in continuum intensity mode:

```
# Use fido to search for an SDO HMI continuum intensity image
# between the two times below
result = Fido.search(a.Time('2013/5/13 15:55', '2013/5/13 15:55:30'),
                     a.Instrument('HMI'), a.vso.Physobs('intensity'))
```

The variable `result` stores the information about the images that match the search criteria. if we feed it to `Fido.fetch`, sunpy will download the image:

```
# Download the file(s) found to the `data` directory
downloaded_files = Fido.fetch(result, path='data/.')
```

We can open the downloaded image using `Map`, like so:

```
# Use Sunpy to open up the image
m = Map(downloaded_files[0])
```

and finally we can see what the map looks like with:

```
# and use sunpy's built-in features to plot it
m.peek()
plt.show()
```

which gives us the following awesome plot!

CHAPTER 5

Simulating a transit

Now that we have an image of the Sun to work with, let's simulate a transit of an Earth-like planet transiting the Sun.

As with before, we'll have a bunch of packages to import things from:

```
import astropy.units as u
from astropy.constants import R_earth, R_sun
from sunpy.map import Map
from stash import simulate_lightcurve
import matplotlib.pyplot as plt
```

The first thing we'll do is open up the image that we downloaded in the previous example:

```
# Load image from the `fetch_and_plot_example.py` script
image = Map('data/hmi_ic_45s_2013_05_13_15_56_15_tai_continuum.fits').data
```

That's the image of the Sun that we'll simulate a transit on top of. Now let's define some characteristics of the planet that will be doing the transiting:

```
# Set up planet parameters
orbital_period = 365 * u.day
semimajor_axis = 1 * u.AU
impact_parameter = 0.2
R_planet = R_earth
R_star = R_sun
```

The impact parameter `b` is defined on [-1, 1], and describes how far from the center of the solar disk, in units of solar radii, the planet appears to cross over. A planet that transits the solar equator has `b=0`. A planet that just barely grazes the tip of the Sun has `b=1`. (For well-aligned planets, you can convert between the solar latitude being occulted and the impact parameter by noting that $b = \sin\theta$, where θ is the latitude.)

Now we call the `simulate_lightcurve` function to simulate a light curve of a transit of this planet on our image, `image`:

```
# Simulate a light curve for that system, return a `LightCurve` object
lc = simulate_lightcurve(image, orbital_period, semimajor_axis,
                         impact_parameter, R_planet, R_star)
```

We can plot the `LightCurve` object using the built-in plot function:

```
# Plot the resulting light curve
lc.plot()

# Show me the plot!
plt.show()
```

and we'll see something like this:

Would you look at that — the planet occulted a starspot, causing the apparent brightness to temporarily increase during the transit, because less flux was missing when the planet was over the spot, compared to when the planet is over the typically bright photosphere. Now we're cooking!

CHAPTER 6

Simulating a bunch of transits

Now this time, let's iterate over impact parameter and see all of the different transit light curves we could get as we vary $b \in [-1, 0]$:

```
# Iterate over a range of impact parameters:
for impact_parameter in np.arange(-0.8, 0, 0.05):
    # Simulate a light curve for that system, return a `LightCurve` object
    lc = simulate_lightcurve(image, orbital_period, semimajor_axis,
                             impact_parameter, R_planet, R_star)

    # Plot the resulting light curve
    lc.plot()
```

You can see that the planet occulted the big starspot at one of the impact parameters that we swept through in the `for` loop.

Python Module Index

S

Index