pyCATCH

Stephan G. Heinemann

Nov 09, 2023

CONTENTS:

Welcome to pyCATCH: the phython implementation of the Collection of Analysis Tools for Coronal Holes (CATCH; Heinemann et al. 2019)

CATCH was originally implemented in SSW IDL (See http://lmsal.com/solarsoft/ssw_install.html and http://www.lmsal.com/solarsoft/ssw_packages_info.html). It is also available on GitHub (https://github.com/sgheinemann/CATCH)

CATCH and pyCATCH were created in order to collect and structure coronal hole identification, extraction and analysis in a handy and fast way without the disadvantages of automatic algorithms. It enables the user to download and process EUV filtergrams (193/195 A) and line-of-sight (Los) magnetograms. It is able to handle data from different spacecraft missions covering the interval from 1996 until now. These include the Solar Dynamics Observatory, the Solar Terrestrial Relations Observatory and the Solar and Heliospheric Observatory.

The user can perform coronal hole boundary detection, extraction and analysis using a manually adjustable intensity threshold. Additionally the user can analyze the underlying photospheric magnetic field.

If you have any comments, suggestions or need help. Please contact the author via E-mail (stephan.heinemann@hmail.at) or leave a request on Github (https://github.com/sgheinemann/CATCH or https://github.com/sgheinemann/pycatch respectively).

You can read the documentation by opening the

```
doc/_build/index.html
```

file.

CHAPTER ONE

GETTING STARTED WITH PYCATCH

This section provides a quick introduction to using pyCATCH.

pyCATCH essentially consists only of the class object pycatch that inclused inbuilt methods for detection and extraction of coronal holes.

It uses a lot of sunpy <https://sunpy.org/> functionalites, thus for advanced usage of pyCATCH, it is suggested to make yourself familiar with it.

1.1 Reading the documentation

You can read the documentation by opening the

```
doc/_build/html/index.html
```

file.

1.2 Installing pyCATCH

For pyCATCH version < 1.0.0: Download or pull pyCATCH repository from GitHub

```
https://github.com/sgheinemann/pycatch
```

Navigate to the directory and install pyCATCH using pip

```
pip install .
```

pyCATCH uses the following packages:

- aiapy
- astropy
- joblib
- matplotlib
- numexpr
- numpy
- opencv-python
- sunpy

- reproject
- scipy

1.3 Initializing pyCATCH

Import the pycatch class with

```
from pycatch.pycatch import pycatch
```

then you can initilize the class

```
ch = pycatch()
```

1.4 Dowloading data

EUV data can be downloaded with

```
ch.download('DATE')
```

and respectively magnetograms, if a EUV map is already loaded

```
ch.download_magnetogram()
```

1.5 Loading data

EUV images and magnetograms can be loaded into the pyCATCH class by

```
ch.load(file='PATH')
ch.load(file='PATH', mag = True )
```

If data was downloaded with the same instance of the pyCATCH class, the path to the downloaded data is stored in the class and the load method can be called without providing the path.

1.6 Calibrating data

Prepare the date for coronal hole extraction

```
ch.calibration()
ch.calibration_mag()
```

The data can rebinned and cutout if desired

```
ch.rebin(ndim=(nx,ny))
ch.cutout(top=(x_max,y_max), bot=(x_min,y_min))
```

1.7 Selecting coronal hole seed point

Select a seed point from an interactive window from where the coronal hole will be grown

```
ch.select(hint=True)
```

Flag hint to highlight dark regions (possible but not necessarily all coronal hole).

1.8 Setting a threshold

pyCATCH features four different options to select a threshold.

- Set the threshold manually

  ```
  ch.set_threshold(Threshold)
  ```

- Use the threshold derived from CATCH statistics (Heinemann et al. 2019)

  ```
  ch.suggest_threshold()
  ```

  It is advised to use this option to get a starting suggestion and then adjust the threshold as needed.

- Select the threshold from solar disk intensity distribution

  ```
  ch.threshold_from_hist()
  ```

- Calculate the coronal hole area and uncertainty as function of intensity and select the boundary to be where the uncertainty is lowest.

  ```
  ch.calculate_curves()
  ch.threshold_from_curves()
  ```

  Warning: This is an advance option and can be slow with high resolution.

1.9 Extracting the coronal hole

The coronal hole can be extracted from the selected seed point and the set threshold

```
ch.extract_ch()
```

1.10 Calculating coronal hole properties

The coronal hole's morphological properties can be calculated with

```
ch.calculate_properties()
```

and the magnetic properties can be calculated with

```
ch.calculate_properties(mag=True)
```

1.11 Plotting and saving coronal hole extractions

The intensity map with the coronal hole boundary and uncertainties overlaid can be displayed with

```
ch.plot_map()
```

Set the 'mag' flag to display the magnetogram instead

```
ch.plot_map(mag=True)
```

Set the 'save' flag to save the images to PDF

```
ch.plot_map(save=True)
ch.plot_map(mag=True,save=True)
```

The properties of the extracted coronal hole can be saved in a text file with

```
ch.print_properties()
```

And the extracted coronal hole 'binary' map can be saved as a fits file with

```
ch.bin2fits()
```

1.12 Saving and loading pyCATCH session

A pyCATCH object can be save to a pickle file

```
ch.save()
```

and restored with

```
ch_loaded = pycatch(restore='PATH')
```

1.13 Example

Example python scripts can be found in the directory

```
examples/
```

CHAPTER

TWO

PYCATCH

2.1 pycatch.pycatch

The *pycatch* class is the primary access point to pyCATCH's functionalities.

`class pycatch.pycatch.`**`pycatch`**`(`*restore=None*, ***kwargs*`)`

Bases: `object`

A Python library for extracting and analyzing coronal holes from solar EUV images and magnetograms.

2.1.1 Attributes

dir
[str] The directory path. Defaults to the user's home directory if not provided.

save_dir
[str] The save directory path. Defaults to the user's home directory if not provided.

map_file
[str] The map file path.

magnetogram_file
[str] The magnetogram file path.

map
[sunpy.map.Map] The loaded and configured EUV map. This map contains both the 2D data array and metadata associated with the EUV observation.

original_map
[sunpy.map.Map] The original EUV map. This map contains the initially loaded EUV observation before any operations.

magnetogram
[sunpy.map.Map] The loaded and configured magnetogram. This map contains both the 2D data array and metadata associated with the magnetic field data.

point
[list of float] Seed point for coronal hole extraction.

curves
[tuple]

- The threshold range.
- The calculated area curves for the coronal hole.

- The uncertainty in the area curves.

threshold
[float] Coronal hole extraction threshold.

type
[str] Placeholder for type information.

rebin_status
[bool] Whether the map was rebinned.

cutout_status
[bool] Whether the map was cutout.

kernel
[int] Size of the circular kernel for morphological operations.

binmap
[sunpy.map.Map] Single 5-level binary map with the coronal hole extraction, where each level represents a different threshold value.

properties
[dict] A dictionary containing the calculate coronal hole properties.

__version__
[str] Version number of pyCATCH

2.1.2 Parameters

dir
[str, optional] Directory for storing and loading data. Default is the home directory.

save_dir
[str, optional] Directory for storing data. Default is the home directory.

map_file
[str, optional] Filepath to EUV/Intensity map, needs to be loadable with sunpy.map.Map(). Default is None.

magnetogram_file
[str, optional] Filepath to magnetogram, needs to be loadable with sunpy.map.Map(). Default is None.

load
[str, optional] Loads a previously saved pyCATCH object from the specified path, which overrides any other keywords. Default is None.

2.1.3 Returns

None

2.1.4 Methods

`bin2fits`(*file=None*, *small=False*, *overwrite=False*)

Save the coronal hole binary map to a FITS file.

Parameters

file
[str, optional] Filepath to save the FITS file. If not provided, a default filename will be generated based on observation metadata. Default is None.

small
[bool, optional] Save a smaller region around the coronal hole. Default is False.

overwrite
[bool, optional] Flag to overwrite the file if it already exists. Default is False.

Returns

None

`calculate_curves`(*verbose=True*)

Calculate area and uncertainty curves as a function of intensity.

Parameters

verbose
[bool, optional] Display warnings. Default is True.

Returns

None

`calculate_properties`(*mag=False*, *align=False*)

Calculate the morphological coronal hole properties from the extracted binary map.

Parameters

mag
[bool, optional] Calculate magnetic properties instead. Default is False.

align
[bool, optional] Call pycatch.calibration_mag() to align with binary map. Default is False.

Returns

None

`calibration(**kwargs)`

Calibrate the intensity image.

Parameters

**** kwargs (SDO/AIA) :**

deconvolve
[bool or numpy.ndarray, optional] Use PSF deconvolution. Default is None. It takes a custom PSF array as input, if True uses aiapy.psf.deconvolve. WARNING: can take about 10 minutes.

register
[bool, optional] Co-register the map. Default is True.

normalize
[bool, optional] Normalize intensity to 1s. Default is True.

degradation
[bool, optional] Correct instrument degradation. Default is True.

alc
[bool, optional] Apply Annulus Limb Correction, Python implementation from Verbeek et al. (2014). Default is True.

cut_limb
[bool, optional] Set off-limb pixel values to NaN. Default is True.

**** kwargs (STEREO/SECCHI) :**

deconvolve
[bool, optional] NOT YET IMPLEMENTED FOR STEREO.

register
[bool, optional] Co-register the map. Default is True.

normalize
[bool, optional] Normalize intensity to 1s. Default is True.

alc
[bool, optional] Apply Annulus Limb Correction, Python implementation from Verbeek et al. (2014). Default is True.

cut_limb
[bool, optional] Set off-limb pixel values to NaN. Default is True.

Returns

None

`calibration_mag`(***kwargs*)

Calibrate the magnetogram.

Parameters

**** kwargs (SDO/HMI)**
[]

rotate
[bool, optional] Rotate the map so that North is up. Default is True.

align
[bool, optional] Align with an AIA map. Default is True.

cut_limb
[bool, optional] Set off-limb pixel values to NaN. Default is True.

Returns

None

`cutout`(*top=(1100, 1100)*, *bot=(-1100, -1100)*)

Cut a subfield of the map (if a magnetogram is loaded, it will also be cut).

Parameters

top
[tuple, optional] Coordinates of the top-right corner. Default is (1100, 1100).

bot
[tuple, optional] Coordinates of the bottom-left corner. Default is (-1100, -1100)).

Returns

None

`download`(*time*, *instr='AIA'*, *wave=193*, *source='SDO'*, ***kwargs*)

Download an EUV map using VSO (Virtual Solar Observatory). It downloads the closest image within +/-1 hour of the time provided.

Parameters

time
[tuple, list, str, pandas.Timestamp, pandas.Series, pandas.DatetimeIndex, datetime.datetime, datetime.date, numpy.datetime64, numpy.ndarray, astropy.time.Time] The time of the EUV image to download. Input needs be parsed by sunpy.time.parse_time()

instrument
[str, optional] The instrument of the EUV image to download. Default is 'AIA'.

source
[str, optional] The source of the EUV image to download. Default is 'SDO'.

wavelength
[int, optional] The wavelength of the EUV image to download (in Angstrom). Default is 193.

**** kwargs:**
Additional keyword arguments passed to sunpy.Fido.search (see sunpy documentation for more information).

Returns

None

Notes

For EIT data: instr='EIT', wave=195, source='SOHO' For STEREO-A data: instr='SECCHI', wave=195, source='STEREO_A' For STEREO-B data: instr='SECCHI', wave=195, source='STEREO_B'

`download_magnetogram`(*cadence=45*, *time=None*, ***kwargs*)

Download a magnetogram matching the EUV image date using VSO (Virtual Solar Observatory). It downloads the closest image within +/- 1 hour ot the time of the EUV image.

Parameters

cadence
[int, optional] Download Line-of-Sight (LOS) magnetogram with the specified cadence in seconds. Default is 45.

time
[tuple, None or list, str, pandas.Timestamp, pandas.Series, pandas.DatetimeIndex, datetime.datetime, datetime.date, numpy.datetime64, numpy.ndarray, astropy.time.Time] The time of the magnetogram to download. Input needs be parsed by sunpy.time.parse_time(), optional Overrides the date of the EUV map and the filepath of the downloaded data is not stored in self.magnetogram_file.

**** kwargs**
[] Additional keyword arguments passed to sunpy.Fido.search (see sunpy documentation for more information).

Returns

None

`extract_ch`(*kernel=None*)

Extract the coronal hole from the intensity map using the selected threshold and seed point. This function outputs a binary map to pycatch.binmap.

Parameters

kernel
[int or None, optional] Size of the circular kernel for morphological operations. Default is None. If None, a kernel size depending on resolution will be used.

Returns

None

`load`(*file=None*, *mag=False*)

Load maps.

Parameters

mag
[bool, optional] Flag to load a magnetogram. Default is False.

file
[str, optional] Filepath to load a specific map. If not set, it loads pycatch.map_file or pycatch.magnetogram_file. Default is None.

Returns

None

`plot_map`(*boundary=True*, *uncertainty=True*, *original=False*, *small=False*, *cutout=None*, *grid=False*, *mag=False*, *fsize=(10, 10)*, *save=False*, *sfile=None*, *overwrite=False*, ***kwargs*)

Display a coronal hole plot.

Parameters

boundary
[bool, optional] Overplot the coronal hole boundary. Default is True.

uncertainty
[bool, optional] Show the uncertainty of the coronal hole boundary. Default is True.

original
[bool, optional] Show the original image. Default is False.

small
[bool, optional] Plot a smaller region around the coronal hole. Default is False. Overrides cutout.

cutout
[list of tuple, optional] Display a cutout around the extracted coronal hole. Format: [(xbot, ybot), (xtop, ytop)]. Default is None.

grid
[bool, optional] Display grid. Default if False.

mag
[bool, optional] Show a magnetogram instead of the coronal hole plot. Default is False.

fsize
[tuple, optional] Set the figure size (in inch). Default is (10, 10).

save
[bool, optional] Save and close the figure. Default is False.

sfile
[str, optional] Filepath to save the image as pdf. If not provided, a default filename will be generated based on observation metadata. Use only in conjunction with save=True. Default is None.

overwrite
[bool, optional] Overwrite the plot if it already exists. Default is False.

**** kwargs**
[keyword arguments] Additional keyword arguments for sunpy.map.Map.plot(). See the sunpy documentation for more information.

Returns

None

`print_properties`(*file=None*, *overwrite=False*)

Save properties to a text file.

Parameters

file
[str, optional] Filepath to save the data. Default is pycatch.dir.

overwrite
[bool, optional] Flag to overwrite the file if it already exists. Default is False.

Returns

None

`rebin`(*ndim=(1024, 1024)*, ***kwargs*)

Rebin maps to a new resolution (if a magnetogram is loaded, it will also be resampled).

Parameters

ndim
[tuple, optional] New dimensions of the map. Default is (1024, 1024)).

**** kwargs**
[] Additional keyword arguments passed to sunpy.map.Map.resample (see sunpy documentation for more information).

Returns

None

`save`(*file=None*, *overwrite=False*, *no_original=True*)

Save a pyCATCH object to a pickle file.

Parameters

file
[str, optional] Filepath to save the object, default is pycatch.dir. Default is None.

overwrite
[bool, optional] Flag to overwrite the file if it already exists. Default is False.

no_original
[bool, optional] Flag to exclude saving the original map to save disk space. Default is True.

Returns

None

`select`(*hint=False*, *fsize=(10, 10)*)

Select a seed point from the intensity map.

Parameters

hint
[bool, optional] If True, highlights possible coronal holes. Default is False.

fsize
[tuple, optional] Set the figure size (in inch). Default is (10, 10).

Returns

None

`set_threshold`(*threshold*, *median=True*, *no_percentage=False*)

Set the coronal hole extraction threshold.

Parameters

threshold
[float] The threshold value.

median
[bool, optional] If True, the input is assumed to be a fraction of the median solar disk intensity. Default is True.

no_percentage
[bool, optional] If True, the input is given as a percentage of the median solar disk intensity. Default is False. This only works in conjunction with median=True.

Returns

None

`suggest_threshold()`

Suggest a coronal hole extraction threshold based on the CATCH statistics (Heinemann et al. 2019).

This function calculates a threshold value using the formula: TH = 0.29 * Im + 11.53 [DN]

where Im is the median solar disk intensity in Data Numbers (DN).

Parameters

None

Returns

None

`threshold_from_curves`(*fsize=(10, 5)*)

Select a threshold for coronal hole extraction from calculated area and uncertainty curves as a function of intensity.

Before using this function, you need to calculate the curves using *pycatch.calculate_curves()*.

Parameters

fsize
[tuple, optional] Set the figure size (in inch). Default is (10, 5).

Returns

None

`threshold_from_hist`(*fsize=(10, 5)*)

Select a threshold for coronal hole extraction from the solar disk intensity histogram.

Parameters

fsize
[tuple, optional] Set the figure size (in inch). Default is (10, 5).

Returns

None

2.2 pycatch.utils

The *utils* package contains utility submodules.

2.2.1 pycatch.utils.calibration Module

The *calibration* module provides calibration functions.

`pycatch.utils.calibration.`**`annulus_limb_correction`**`(`*map*`)`

Apply annulus limb correction to a solar map.

This function performs annulus limb correction on a solar map following the method described in Verbeek et al. (2014). Transferred from IDL to Python 3 by S.G. Heinemann, June 2022

Parameters

map
[sunpy.map.Map] The input solar map to which the limb correction will be applied.

Returns

sunpy.map.Map
A solar map with annulus limb correction applied.

`pycatch.utils.calibration.`**`calibrate_aia`**`(`*map*, *register=True*, *normalize=True*, *deconvolve=None*, *alc=True*, *degradation=True*, *cut_limb=True*`)`

Calibrate and preprocess an AIA (Atmospheric Imaging Assembly) map.

This function performs various calibration and preprocessing steps on an AIA map to prepare it for further analysis.

Parameters

map
[sunpy.map.Map] The input AIA map to be calibrated and preprocessed.

register
[bool, optional] Whether to perform image registration. Default is True.

normalize
[bool, optional] Whether to normalize the exposure. Default is True.

deconvolve
[bool or None or numpy.ndarray, optional] Whether to perform PSF deconvolution. If set to True, deconvolution with aiapy.psf.deconvolve is applied. If custom PSF array is given, it uses this array instead. If set to False, it is not applied.

alc
[bool, optional] Whether to perform annulus limb correction. Default is True.

degradation
[bool, optional] Whether to correct for instrument degradation. Default is True.

cut_limb
[bool, optional] Whether to cut the limb of the solar disk. Default is True.

Returns

sunpy.map.Map
A calibrated and preprocessed AIA map ready for analysis.

`pycatch.utils.calibration.`**`calibrate_hmi`**(*map*, *intensity_map*, *rotate=True*, *align=True*, *cut_limb=True*)

Calibrate and preprocess an HMI (Helioseismic and Magnetic Imager) map.

This function performs various calibration and preprocessing steps on an HMI map to prepare it for further analysis.

Parameters

map
[sunpy.map.Map] The input HMI map to be calibrated and preprocessed.

intensity_map
[sunpy.map.Map] An intensity map to align the HMI map with.

rotate
[bool, optional] Whether to rotate the HMI map to have north up. Default is True.

align
[bool, optional] Whether to align the HMI map with an intensity map. Default is True.

cut_limb
[bool, optional] Set off-limb pixel values to NaN. Default is True.

Returns

sunpy.map.Map
A calibrated and preprocessed HMI map ready for analysis.

`pycatch.utils.calibration.`**`calibrate_stereo`**(*map*, *register=True*, *normalize=True*, *deconvolve=None*, *alc=True*, *cut_limb=True*)

Calibrate and preprocess a STEREO EUV (Solar TErrestrial RElations Observatory) map.

This function performs various calibration and preprocessing steps on a STEREO map to prepare it for further analysis.

Parameters

map
[sunpy.map.Map] The input STEREO map to be calibrated and preprocessed.

register
[bool, optional] Whether to perform map rotation to register the map. Default is True.

normalize
[bool, optional] Whether to normalize the exposure. Default is True.

deconvolve
[bool or None or numpy.ndarray, optional] Whether to perform PSF deconvolution. If set to True, deconvolution with aiapy.psf.deconvolve is applied. If custom PSF array is given, it uses this array instead. If set to False, it is not applied.

alc
[bool, optional] Whether to perform annulus limb correction. Default is True.

cut_limb
[bool, optional] Set off-limb pixel values to NaN. Default is True.

Returns

sunpy.map.Map
A calibrated and preprocessed STEREO map ready for analysis.

2.2.2 pycatch.utils.ch_mapping Module

The *ch_mapping* module provides map operation functions.

`pycatch.utils.ch_mapping.`**`calc_area_curves`**(*map*, *th*, *kernel*, *seed*, *minval*, *coreg*)

Wrapper to calculate area curves for a coronal hole in a solar map.

Parameters

map
: [sunpy.map.Map] The input solar map containing coronal hole data.

th
: [float] The threshold value for coronal hole extraction.

kernel
: [int or None] The size of the circular kernel for morphological operations.

seed
: [tuple] The seed point coordinates (lon, lat) for coronal hole extraction.

minval
: [float] The minimum threshold value to consider for area calculation.

coreg
: [sunpy.map.Map] The coregistered solar map for area calculation.

Returns

numpy.ndarray
: The area curves for the coronal hole as function of intensity.

`pycatch.utils.ch_mapping.`**`catch_calc`**(*binmaps*, *binary=True*)

Calculate coronal hole properties from a list of binary maps.

This function calculates various properties of coronal holes, such as area, center of mass, extent, and their uncertainties, from a list of binary maps representing different levels of coronal hole segmentation.

Parameters

binmaps
: [list of sunpy.map.Map] A list of binary maps representing different levels of coronal hole segmentation.

binary
: [bool, optional] Indicates whether the input binary maps are binary (Default is True).

Returns

sunpy.map.Map
: A binary map representing the combined coronal hole regions.

float
: The mean area of coronal holes.

float
: The uncertainty of the mean area of coronal holes.

numpy.ndarray
: An array containing the mean center of mass (lon, lat) of coronal holes.

numpy.ndarray
: An array containing the uncertainty of the mean center of mass (lon, lat) of coronal holes.

numpy.ndarray
: An array containing the mean extent (lon1, lon2, lat1, lat2) of coronal holes.

numpy.ndarray
An array containing the uncertainty of the mean extent (lon1, lon2, lat1, lat2) of coronal holes.

`pycatch.utils.ch_mapping.`**`catch_mag`**`(`*binmaps*, *magmap*`)`

Wrapper to calculate magnetic properties of coronal holes from a list of binary maps and a magnetic field map.

This function calculates various magnetic properties of coronal holes, including signed and unsigned mean magnetic flux density, signed and unsigned magnetic flux, and flux balance, from a list of binary maps representing different levels of coronal hole segmentation and a magnetic field map.

Parameters

binmaps
[list of sunpy.map.Map] A list of binary maps representing different levels of coronal hole segmentation.

magmap
[sunpy.map.Map] The magnetic field map.

Returns

float
The mean signed magnetic flux density of coronal holes.

float
The uncertainty of the mean signed magnetic flux density of coronal holes.

float
The mean unsigned magnetic flux density of coronal holes.

float
The uncertainty of the mean unsigned magnetic flux density of coronal holes.

float
The mean signed magnetic flux of coronal holes.

float
The uncertainty of the mean signed magnetic flux of coronal holes.

float
The mean unsigned magnetic flux of coronal holes.

float
The uncertainty of the mean unsigned magnetic flux of coronal holes.

float
The mean flux balance of coronal holes.

float
The uncertainty of the mean flux balance of coronal holes.

`pycatch.utils.ch_mapping.`**`catch_uncertainty`**`(`*x*`)`

Calculate the uncertainty of a given data array.

This function computes the uncertainty of a given data array using the maximum absolute deviation from the mean.

Parameters

x
[numpy.ndarray] The input data array for which uncertainty is calculated.

Returns

float
The uncertainty value calculated as the maximum absolute deviation from the mean.

`pycatch.utils.ch_mapping.`**`ch_area`**`(`*map*, *coreg=None*, *binary=False*`)`

Calculate the area of a coronal hole in a solar map.

Parameters

map
[sunpy.map.Map] The input solar map containing coronal hole data.

coreg
[numpy.ndarray, optional] An optional curvature correction factor for the map. If not provided, it will be computed internally.

binary
[bool, optional] If True, the map is treated as binary data, where coronal hole pixels have a value of 1. If False, the map is thresholded to binary data. Default is False.

Returns

float
The calculated area of the coronal hole in 10^10 km^2.

`pycatch.utils.ch_mapping.`**`ch_flux`**`(`*binmap*, *magmap*, *coreg=[0]*`)`

Calculate magnetic properties of a coronal hole region from a binary map and a magnetic field map.

This function calculates various magnetic properties of a coronal hole region, including signed mean magnetic flux density, unsigned mean magnetic flux density, signed magnetic flux, and unsigned magnetic flux, based on a binary map representing the coronal hole region and a magnetic field map.

Parameters

binmap
[sunpy.map.Map] A binary map representing the coronal hole region.

magmap
[sunpy.map.Map] The magnetic field map.

coreg
[list of float, optional] An optional curvature correction factor for the map. If not provided, it will be computed internally.

Returns

float
The mean signed magnetic flux density of the coronal hole region.

float
The mean unsigned magnetic flux density of the coronal hole region.

float
The signed magnetic flux of the coronal hole region.

float
The unsigned magnetic flux of the coronal hole region.

`pycatch.utils.ch_mapping.`**`curve_corr`**`(`*map*`)`

Compute a correction factor for curvature in a solar map.

Parameters

map
[sunpy.map.Map] The input solar map for which the correction factor is computed.

Returns

numpy.ndarray
A 2D array representing the correction factor for curvature in the solar map.

`pycatch.utils.ch_mapping.`**`cutout`**`(`*map*, *top*, *bot*`)`

Create a cutout region from a solar map based on top and bottom coordinates.

This function generates a cutout region from the input solar map (*map*) using specified top and bottom coordinates.

Parameters

map
[sunpy.map.Map] The input solar map from which the cutout region will be created.

top
[tuple] A tuple containing the top-right corner coordinates (x, y) of the cutout region in arcseconds.

bot
[tuple] A tuple containing the bottom-left corner coordinates (x, y) of the cutout region in arcseconds.

Returns

sunpy.map.Map
A cutout region of the input solar map based on the specified coordinates.

`pycatch.utils.ch_mapping.`**`extract_ch`**`(`*map*, *thr*, *kernel*, *seed*`)`

Extract a coronal hole from a solar map based on a threshold and seed point.

Parameters

map
[sunpy.map.Map] The input solar map containing coronal hole data.

thr
[float] The threshold value for coronal hole extraction.

kernel
[int or None] The size of the circular kernel for morphological operations. If None, the kernel size is automatically determined based on map resolution.

seed
[tuple] The seed point coordinates (x, y) for coronal hole extraction.

Returns

sunpy.map.Map
A “binary” map containing the extracted coronal hole region.

`pycatch.utils.ch_mapping.`**`from_5binmap`**`(`*binmap*`)`

Split a 5-level binary map into multiple binary maps.

Parameters

binmap
[sunpy.map.Map] A 5-level binary map containing multiple threshold levels.

Returns

list of sunpy.map.Map
A list of binary maps, one for each threshold level extracted from the input 5-level binary map.

`pycatch.utils.ch_mapping.`**`get_curves`**`(`*map*, *seed*, *kernel=None*, *upper_lim=False*, *cores=8*`)`

Top-level wrapper to calculate coronal hole area curves for a range of threshold values.

This function calculates coronal hole area curves for a range of threshold values, optionally considering an upper limit. It also computes uncertainty in the area curves. The function uses parallel processing to speed up computation.

Parameters

map
[sunpy.map.Map] The input solar map containing coronal hole data.

seed
[tuple] The seed point coordinates (x, y) for coronal hole extraction.

kernel
[int or None, optional] The size of the circular kernel for morphological operations. Default is None.

upper_lim
[int or False, optional] The upper limit for the threshold range. If False, the limit is calculated based on the median value of the solar map data. Default is False.

cores
[int, optional] The number of CPU cores to use for parallel processing. Default is 8.

Returns

tuple
A tuple containing three arrays: - The threshold range. - The calculated area curves for the coronal hole. - The uncertainty in the area curves.

`pycatch.utils.ch_mapping.`**`get_intensity`**`(`*binmaps*, *map*`)`

Calculate intensity properties of coronal holes from a list of binary maps.

This function calculates the mean and median intensity properties of coronal holes from a list of binary maps representing different levels of coronal hole segmentation.

Parameters

binmaps
[list of sunpy.map.Map] A list of binary maps representing different levels of coronal hole segmentation.

map
[sunpy.map.Map] The original intensity map.

Returns

float
The mean intensity of coronal holes.

float
The uncertainty of the mean intensity of coronal holes.

float
The median intensity of coronal holes.

float
The uncertainty of the median intensity of coronal holes.

`pycatch.utils.ch_mapping.`**`make_circle`**`(`*s*`)`

Generate a binary circular mask.

This function generates a binary circular mask with a specified size, where the circle is centered within the mask.

Parameters

s
: [int] The size of the circular mask (side length).

Returns

numpy.ndarray
: A binary circular mask with the specified size.

`pycatch.utils.ch_mapping.`**`min_picker`**`(`*seed*, *data*`)`

Find the minimum value in a local neighborhood around a seed point in a data array.

This function identifies the minimum value within a 5x5 local neighborhood centered around a seed point (xs, ys) in the input data array.

Parameters

seed
: [tuple] The seed point coordinates (xs, ys) as a tuple.

data
: [array-like] The input data array in which to search for the minimum value.

Returns

float
: The minimum value within the local neighborhood.

`pycatch.utils.ch_mapping.`**`to_5binmap`**`(`*binmaps*`)`

Combine multiple binary maps into a single 5-level binary map.

Parameters

binmaps
: [list of sunpy.map.Map] A list of binary maps to be combined.

Returns

sunpy.map.Map
: A single 5-level binary map where each level represents a different threshold value.

2.2.3 pycatch.utils.extensions Module

The *extensions* module provides additional functions.

`pycatch.utils.extensions.`**`congrid`**`(`*a*, *newdims*, *method='linear'*, *centre=False*, *minusone=False*`)`

Resample an array to new dimension sizes using various interpolation methods.

Parameters

a
[array-like] The input array to be resampled.

newdims
[tuple] The new dimensions to which the array should be resampled.

method
[str, optional] The interpolation method to use. Default is 'linear'.

centre
[bool, optional] Whether interpolation points are at the centers of the bins. Default is False.

minusone
[bool, optional] Whether to prevent extrapolation one element beyond the bounds of the input array. Default is False.

Returns

array-like
The resampled array with the specified dimensions and interpolation method applied.

Notes

This function is adapted from IDL's *congrid* routine. Arbitrary resampling of source array to new dimension sizes. Currently only supports maintaining the same number of dimensions. To use 1-D arrays, first promote them to shape (x,1). Uses the same parameters and creates the same co-ordinate lookup points as IDL"s congrid routine, which apparently originally came from a VAX/VMS routine of the same name.

method: neighbour - closest value from original data nearest and linear - uses n x 1-D interpolations using scipy.interpolate.interp1d (see Numerical Recipes for validity of use of n 1-D interpolations) spline - uses ndimage.map_coordinates

centre: True - interpolation points are at the centres of the bins False - points are at the front edge of the bin

minusone: For example- inarray.shape = (i,j) & new dimensions = (x,y) False - inarray is resampled by factors of (i/x) * (j/y) True - inarray is resampled by(i-1)/(x-1) * (j-1)/(y-1) This prevents extrapolation one element beyond bounds of input array.

`pycatch.utils.extensions.`**`find_nearest`**`(`*array*, *value*`)`

Find the index of the nearest value in an array to a specified value.

Parameters

array
[array-like] The input array in which to find the nearest value.

value
[float] The value to which the nearest element in the array is sought.

Returns

int
The index of the nearest value in the array.

`pycatch.utils.extensions.`**`get_extent`**`(`*map*`)`

Calculate the extent of a coronal hole in Helioprojective Cartesian (HPC) coordinates.

Parameters

map
[sunpy.map.Map] A SunPy map object for which the extent needs to be calculated.

Returns

tuple
A tuple containing two tuples representing the lower-left and upper-right corners of the map's extent in HPC coordinates. The format of the outer tuple is ((x_min, y_min), (x_max, y_max)), where: (x_min, y_min) represents the HPC coordinates of the lower-left corner. (x_max, y_max) represents the HPC coordinates of the upper-right corner.

`pycatch.utils.extensions.`**`init_props`**`()`

Initialize a dictionary of properties and their units for pyCATCH.

Returns

dict
A dictionary where keys are property abbreviations, and values are tuples containing the full property name and its unit.

`pycatch.utils.extensions.`**`median_disk`**`(`*map*`)`

Calculate the median value within the solar disk region of a map.

Parameters

map
: [sunpy.map.Map] The input solar map.

Returns

float
: The median value of the data within the solar disk.

`pycatch.utils.extensions.printtxt(file, pdict, names, version)`

Write property data to a text file.

This function writes property data to a text file with a specific format. It includes the version number, property names, and units as headers followed by the property values.

Parameters

file : str The filepath to the output text file. pdict : dict A dictionary containing property data, where keys correspond to property abbreviations. names : dict A dictionary mapping property abbreviations to tuples containing full property names and units. version : str The version number of the pyCATCH software.

Returns

None

2.2.4 pycatch.utils.plot Module

The *plot* module provides plotting functions.

`class pycatch.utils.plot.SnappingCursor(fig, ax, line, line2=None, names=['y'], xe=None, ye=None)`

Bases: `object`

A cross-hair cursor that snaps to the data point of a line, which is closest to the x position of the cursor.

For simplicity, this assumes that x values of the data are sorted.

Parameters

fig
: [matplotlib.figure.Figure] The matplotlib figure to which the cursor is attached.

ax
: [matplotlib.axes.Axes] The matplotlib axes to which the cursor is attached.

line
: [matplotlib.lines.Line2D] The line for which the cursor will snap to data points.

line2
: [matplotlib.lines.Line2D, optional] An optional second line for which the cursor can snap to data points.

names
: [list of str, optional] A list of names for the y-values displayed in the cursor's tooltip.

xe
[array-like, optional] An array of x-values associated with the data points.

ye
[array-like, optional] An array of y-values associated with the data points.

Attributes

location
[float] The x-value of the currently snapped data point.

Methods

on_mouse_click(event)
Handles mouse clicks to capture the cursor's location.

on_mouse_move(event)
Handles mouse movement to snap the cursor to the nearest data point.

on_draw(event)
Handles drawing events to update the cursor's background.

`pycatch.utils.plot.`**`get_point_from_map`**`(`*map*, *hint*, *fsize*`)`

Display a solar map and allow the user to interactively select a point on the map.

Parameters

map
[sunpy.map.GenericMap] The solar map to be displayed.

fsize
[tuple of int] The size of the figure (width, height) in inches.

Returns

point
[list of float] A list containing the coordinates (x, y) of the selected point on the solar map.

`pycatch.utils.plot.`**`get_thr_from_curves`**`(`*map*, *curves*, *fsize*`)`

Interactively get a threshold value from a plot of coronal hole area curves.

Parameters

map
[sunpy.map.GenericMap] The solar map for which the threshold will be determined.

curves
[tuple] A tuple containing the data for plotting the curves, where *curves[0]* is the x-axis data, *curves[1]* is the y-axis data for the first curve, and *curves[2]* is the y-axis data for the second curve.

fsize
[tuple of int] The size of the figure (width, height) in inches.

Returns

thr
[float] The threshold value determined interactively from the plot.

`pycatch.utils.plot.`**`get_thr_from_hist`**`(`*map*, *fsize*`)`
Interacticely get a threshold value from a histogram of solar disk data.

Parameters

map
[sunpy.map.GenericMap] The solar map for which the threshold will be determined.

fsize
[tuple of int] The size of the figure (width, height) in inches.

Returns

thr
[float] The threshold value determined interactively from the histogram.

`pycatch.utils.plot.`**`plot_map`**`(`*map*, *bmap*, *boundary*, *uncertainty*, *fsize*, *save*, *spath*, *grid*, ***kwargs*`)`
Plot a solar map with optional boundaries and uncertainty overlays.

Parameters

map
[sunpy.map.GenericMap] The solar map to be plotted.

bmap
[sunpy.map.GenericMap or None] A binary mask map for boundaries or uncertainty, or None if not used.

boundary
[bool] Whether to plot boundaries on the map.

uncertainty
[bool] Whether to overlay uncertainty information on the map.

fsize
[tuple of int] The size of the figure (width, height) in inches.

save
[bool] Whether to save the figure to a file.

spath
[str] The path to save the figure if *save* is True.

****kwargs**
Additional keyword arguments to pass to the *sunpy.map.Map.plot* function.

Returns

None

CHAPTER

THREE

CHANGELOG

3.1 0.2.1 (Novemeber 9, 2023)

- **Bug fixing**
  - fixed a bug where importing pycatch would not find the _version.py file
  - fixed some path issue in the initialization of the home path
  - fixed what version was output in the .print_properties() function
  - fixed some function descriptions
  - fixed a problem where the .print_properties() function would not write a file if no magnetic properties were calculated.
  - fixed an issue with where possibly additional windows open when trying to select a seed point
- **Minor changes**
  - changed the keyword order in the .load() routine to make it more intuitive
  - added aiapy to the list of required packages
  - switched default for .plot_map() from small from True to False
  - added a pdf version of the documentation (User_Manual.pdf that can be found on the github page)

3.2 0.2.0 (September 2023)

- Initial functional beta release (beta - testing)

3.3 0.1.0 (September 2023)

- Initial alpha build

CHAPTER FOUR

ACKNOWLEDGING OR CITING PYCATCH

If you use pyCATCH in your scientific work, we would appreciate citing and acknowledging it in your publications.

4.1 DISCLAIMER

If you use pyCATCH versions < 1.0.0 (unpublished version), please get in contact with Stephan G. Heinemann (stephan.heinemann@hmail.at) before publication.

4.1.1 Citing pyCATCH in Publications

Please add the following line within your methods, conclusion or acknowledgements sections:

This research used version X.Y.Z of the pyCATCH open source software package.

The package citation should be to the CATCH paper, with the disclaimer that the python implementation was used.

```
@ARTICLE{2019SoPh..294..144H,
       author = {{Heinemann}, Stephan G. and {Temmer}, Manuela and {Heinemann}, Niko and
↪{Dissauer}, Karin and {Samara}, Evangelia and {Jer{\v{c}}i{\'c}}, Veronika and
↪{Hofmeister}, Stefan J. and {Veronig}, Astrid M.},
        title = "{Statistical Analysis and Catalog of Non-polar Coronal Holes Covering␣
↪the SDO-Era Using CATCH}",
      journal = {\solphys},
     keywords = {Coronal holes, Magnetic fields, Photosphere, Solar cycle, Observations,␣
↪Astrophysics - Solar and Stellar Astrophysics},
         year = 2019,
        month = oct,
       volume = {294},
       number = {10},
          eid = {144},
        pages = {144},
          doi = {10.1007/s11207-019-1539-y},
archivePrefix = {arXiv},
       eprint = {1907.01990},
 primaryClass = {astro-ph.SR},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2019SoPh..294..144H},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER

FIVE

REFERENCES

Heinemann, S.G., Temmer, M., Heinemann, N., Dissauer, K., Samara, E., Jerčić, V., Hofmeister, S.J., Veronig, A.M.: 2019, Statistical analysis and catalog of non-polar coronal holes covering the SDO-era using CATCH. Solar Phys. 294, 144.

PYTHON MODULE INDEX

INDEX