craterpy

Release 0.5.0

Christian J. Tai Udovicic

May 22, 2021

USAGE

1 Example **3**

2 Installation **7**

3 Dependencies **9**

4 Documentation **11**

5 Contributing **13**
5.1 Bug Reporting and Feature Requests 13
5.2 Contributing Directly 13

6 Citing craterpy **15**

7 Contact **17**
7.1 Craterpy API Documentation 17

Python Module Index **27**

Index **29**

Craterpy simplifies the extraction and statistical analysis of impact craters in planetary datasets. It can:

- work with tables of crater data in Python (using pandas)
- load and manipulate planetary image data in Python (using rasterio)
- extract, mask, filter, and compute stats on craters located in planetary imagery
- plot crater regions of interest

Craterpy currently only supports simple cylindrical images and requires you to provide a table of crater locations and sizes (e.g. it isn't a crater detection program). See the example below!

Note: *Craterpy is in alpha. We appreciate bug reports and feature requests on the issues board.*

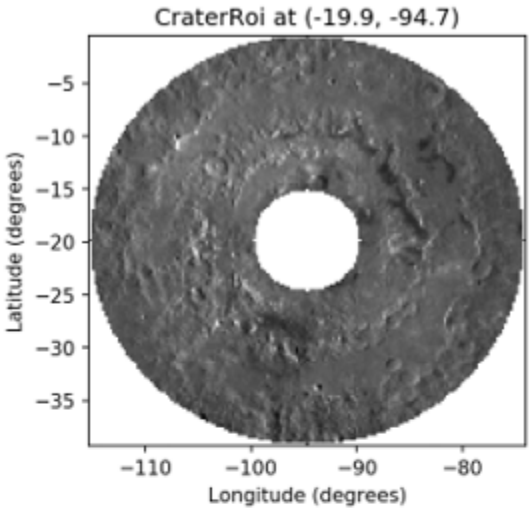
CHAPTER

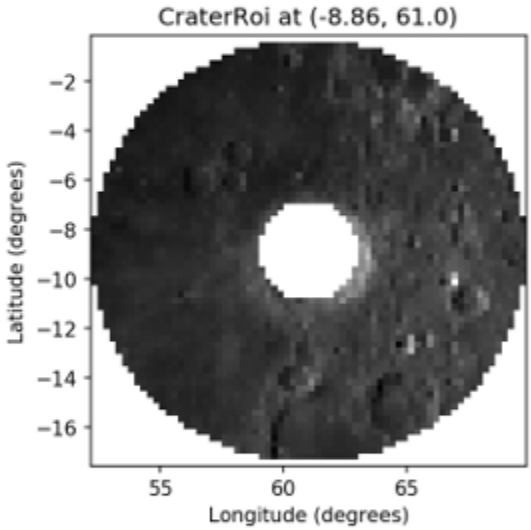
ONE

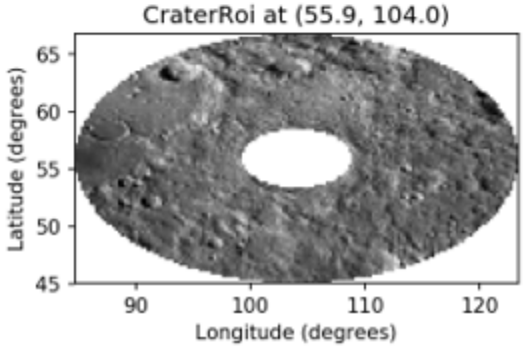
EXAMPLE

Craterpy in action:

```
import pandas as pd
from craterpy import dataset, stats
df = pd.DataFrame({'Name': ["Orientale", "Langrenus", "Compton"],
                   'Lat': [-19.9, -8.86, 55.9],
                   'Lon': [-94.7, 61.0, 104.0],
                   'Rad': [147.0, 66.0, 82.3]})
moon = dataset.CraterpyDataset("moon.tif")
stat_df = cs.ejecta_stats(df, moon, 4, ['mean', 'median', 'std'], plot=True)
```








```
stats_df.head()
```

	Lat	Lon	Name	Rad	mean	median	std
0	-19.90	-94.7	Orientale	147.0	57.540698	56.5	16.994025
1	-8.86	61.0	Langrenus	66.0	72.474790	71.0	17.029801
2	55.90	104.0	Compton	82.3	103.119253	101.0	24.475009

New users should start with the Jupyter notebook tutorial for typical usage with examples. See also craterpy documentation on Read the Docs.

Note: This package currently **only accepts image data in simple-cylindrical (Plate Caree) projection**. If your data is in another projection, please reproject it to simple-cylindrical before importing it with craterpy. If you would like add reprojection functionality to craterpy, consider Contributing.

CHAPTER

TWO

INSTALLATION

With pip:

```
pip install craterpy
python -c "import craterpy; print(craterpy.__version__)"
```

In a new conda environment:

```
# Create and activate a new conda environment called "craterpy"
conda create -n craterpy python=3.9
conda activate craterpy

# Install craterpy with pip
pip install craterpy
python -c "import craterpy; print(craterpy.__version__)"
```

With git and poetry (for latest version & development):

```
# Clone this repository
$ cd ~
$ git clone https://github.com/cjtu/craterpy.git

# Enter the repository
$ cd craterpy

# Configure poetry
poetry config virtualenvs.create true --local
poetry config virtualenvs.in-project true --local

# Install craterpy with poetry
$ poetry install

# Check installation
poetry version

# Either open a Jupyter server
$ poetry run jupyter notebook

# Or activate the venv from your Python editor of choice
# The venv is path is ~/craterpy/.venv/bin/python
```

Trouble installing craterpy? Let us know on the issues board.

CHAPTER

THREE

DEPENDENCIES

Craterpy requires python >3.7 and is tested on Ubuntu and OS X. If you would like to use craterpy on Windows, we recommend getting the Windows Subsystem for Linux (WSL) and installing it from the bash shell.

It's core dependencies are:

- rasterio
- pandas
- numpy
- matplotlib

CHAPTER

FOUR

DOCUMENTATION

Full API documentation is available at readthedocs.

CHAPTER

FIVE

CONTRIBUTING

There are two major ways you can help improve craterpy:

5.1 Bug Reporting and Feature Requests

You can report bugs or request new features on the issues board.

5.2 Contributing Directly

Want to fix a bug / implement a feature / fix some documentation? We welcome pull requests from all new contributors! You (yes you!) can help us make craterpy as good as it can be! See CONTRIBUTING.rst for details on how to get started - first time GitHub contributors welcome - and encouraged!

CHAPTER

SIX

CITING CRATERPY

Craterpy is MIT Licenced and is free to use with attribution. Citation information can be found here.

CHAPTER

SEVEN

CONTACT

If you have comments/question/concerns or just want to get in touch, you can email Christian at cj.taiudovicic@gmail.com or follow @TaiUdovicic on Twitter.

7.1 Craterpy API Documentation

7.1.1 CraterpyDataset object

class craterpy.dataset.**CraterpyDataset**(*dataset*, *nlat=None*, *slat=None*, *wlon=None*, *elon=None*, *radius=None*, *xres=None*, *yres=None*)

The CraterpyDataset reads in images with the rasterio dataset.

If the input file is georeferenced, the geographical bounds and transform will be read automatically. Otherwise, all attributes must be passed in the constructor.

Inherits all attributes and methods of an open rasterio.DatasetReader (see https://rasterio.readthedocs.io/en/latest/quickstart.html).

nlat, slat, wlon, elon
North, south, west, and east bounds of dataset [degrees].

Type int or float

radius
Radius of the planetary body [km].

Type int or float

ppd
Resolution of dataset in [pixels per degree].

Type int or float

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'craterpy', 'data')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> ds = CraterpyDataset(dsfile, radius=1737)
>>> ds.get_roi(-27.6, -27.0, 80.5, 81.1).shape
(2, 2)
```

calc_mpp(*lat=0*)

Return the ground resolution in meters/pixel at the given latitude.

Due to stretching towards the poles in the simple-cylindrical projection, mpp resolution is latitude-dependent.

Parameters **lat** (*int or float*) – Latitude (Default is 0, the equator).

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'craterpy', 'data')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> ds = CraterpyDataset(dsfile, radius=1737)
>>> '{:.1f}'.format(ds.calc_mpp())
'7579.1'
>>> '{:.1f}'.format(ds.calc_mpp(50))
'4871.7'
```

inbounds(*lat*, *lon*)

Return True if (lat, lon) point in Dataset bounds.

Parameters

- **lat** (*int or float*) – Latitude [degrees].
- **lon** (*int or float*) – Longitude [degrees].

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'craterpy', 'data')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> ds = CraterpyDataset(dsfile, nlat=20, slat=0, wlon=10, elon=20)
>>> ds.inbounds(10, 15)
True
>>> ds.inbounds(10, 200)
False
>>> ds = CraterpyDataset(dsfile, nlat=20, slat=0, wlon=-180, elon=180)
>>> ds.inbounds(10, 15)
True
>>> ds.inbounds(10, 200)
True
```

is_global()

Check if dataset has 360 degrees of longitude.

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'craterpy', 'data')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> ds = CraterpyDataset(dsfile, radius=1737)
>>> ds.is_global()
True
```

get_roi(*minlon*, *maxlon*, *minlat*, *maxlat*)

Return numpy array of data specified by its geographical bounds

Parameters

- **minlon** (*int or float*) – Western longitude bound [degrees].
- **maxlon** (*int or float*) – Eastern longitude bound [degrees].
- **minlat** (*int or float*) – Southern latitude bound [degrees].
- **maxlat** (*int or float*) – Northern latitude bound [degrees].

Returns **roi** – Numpy array specified by extent given.

Return type numpy 2D array

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'craterpy', 'data')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> ds = CraterpyDataset(dsfile, radius=1737)
>>> ds.get_roi(-27.6, -27.0, 80.5, 81.1).shape
(2, 2)
```

7.1.2 CraterpyRoi object

class craterpy.roi.**CraterRoi**(*cds*, *lat*, *lon*, *rad*, *wsize=1*, *plot=False*)

The CraterRoi contains a region of interest of image data for a crater.

The CraterRoi reads in a 2D numpy array of data from the CraterpyDataset cds and stores it in the roi attribute. The roi is centered on lat, lon of the specified crater, and extends to rad*wsize from the center.

cds

CraterDataset to read roi from.

Type *CraterpyDataset*

lat, lon, radius

Center latitude and longitude of the crater and this roi [degrees].

Type int or float

radius

Radius of the crater [km].

Type int or float

wsize

Size of window around crater [crater radii].

Type int or float

roi
2D numpy array region of interest centered on crater.

Type numpy.ndarray

extent
North lat, south lat, west lon, and east lon bounds of roi [degrees].

Type list of float

Examples

```
>>> import os.path as p
>>> datadir = p.join(p.dirname(p.abspath('__file__')), 'examples')
>>> dsfile = p.join(datadir, 'moon.tif')
>>> cds = CraterpyDataset(dsfile, radius=1737)
>>> croi = CraterRoi(cds, -27.2, 80.9, 207)  # Humboldt crater
```

filter(*vmin=- inf*, *vmax=inf*, *strict=False*, *nodata=nan*)
Filter roi to the inclusive range [vmin, vmax].

You can specify only vmin or vmax if only a lower bound/upper bound is required. Set strict to True to exclude vmin and vmax, i.e. keep data in the exclusive interval (vmin, vmax). The nodata value replaces filtered pixels. It defaults to np.nan.

Parameters

- **vmin** (*int or float*) – Minimum value to keep (default -inf, no lower bound).
- **vmax** (*int or float*) – Maximum value to keep (default inf, no upper bound).
- **strict** (*bool*) – Keep values strictly greater and strictly less than vmin, vmax (default False).
- **nodata** (*int or float*) – Number to fill in filtered values (default np.nan).

Examples

```
>>> croi.filter(0, 1)  # keep data in range (0 < data < 1)
>>> croi.filter(0, 1, True)  # keep data in range ( <= data <= 1)
>>> croi.filter(vmax=20)  # keep data < 20
```

mask(*mask*, *outside=False*, *fillvalue=nan*)
Fill pixels in bool mask from masking.py with fillvalue.

Parameters

- **mask** (*2D array*) – Mask of this roi from masking.py
- **outside** (*bool*) – Mask outside the area in mask (default False).
- **fillvalue** (*int or float*) – Number to fill in masked values (default np.nan).

plot(**args*, ***kwargs*)
Plot this CraterRoi. See plotting.plot_CraterRoi()

save(*fname*)
Save roi to file given by fname

7.1.3 Craterpy stats Module

Compute stats on CraterRoi objects.

craterpy.stats.**ejecta_profile_stats**(*df*, *cds*, *ejrad=2*, *rspacing=0.25*, *stats=None*, *plot_roi=False*, *vmin=None*, *vmax=None*, *strict=False*, *savepath=None*, *timer=False*, *dt=60*)

Compute stat profiles across ejecta blankets by computing stats in a series of concentric rings beginning at the crater rim and extending to ejrad crater radii from the crater centre. Each ring is rspacing thick and the crater floor is excluded. Output is a dictionary of cdf.index values to pandas.DataFrame of stats containing the computed stats as columns and ring radius as rows.

Returns **stat_df** – Dictionary of stats with cdf index as keys and pandas.DataFrame of statistics as values (see example).

Return type dict of (index: pandas.DataFrame)

Example

```
>>> compute_ejecta_profile_stats(my_df, my_cds, 2, 0.25, stats=['mean',
                                  'median','maximum'],savepath='/tmp/')
```

/tmp/index1_ejecta_stats.csv radius,mean,median,maximum 1, 55, 45, 70 1.25, 50, 40, 65 1.5, 35, 35, 45 1.75, 30, 33, 42

craterpy.stats.**compute_stats**(*df*, *cds*, *stats=None*, *plot=False*, *save=False*, *prefix=''*, *vmin=- inf*, *vmax=inf*, *strict=False*, *maskfunc=None*, *mask_out=False*, *ejrad=None*, *buffer=1*, *verbosity=1*)

Computes stats on craters in df with image data from cds.

craterpy.stats.**crater_stats**(*df*, *cds*, *stats=None*, *plot=False*, *vmin=- inf*, *vmax=inf*, *strict=False*, *verbosity=1*)

Computes stats on all craters in df using image data from cds

Crater latitude, longitude, and radius are read in from df. All stats are computed assuming a circular crater centered on (lat, lon). Stats must be present in quickstats.py.

Parameters

- **df** (*pandas.DataFrame object*) – Contains the crater locations and sizes to locate ROIs in aceDS. Also form the basis of the returned DataFrame
- **cds** (*CraterpyDataset object*) – CraterpyDataset of image data used to compute stats.
- **stats** (*Array-like of str*) – Indicates the stat names to compute. Stat functions must be defined in acestats.py. Default: all stats in acestats.py.
- **plot** (*bool*) – Plot the ejecta ROI.
- **vmin** (*int, float*) – The minimum valid image pixel value. Filter all values lower.
- **vmax** (*int, float*) – The maximum valid image pixel value. Filter all values higher.
- **strict** (*bool*) – How strict the (vmin, vmax) range is. If true, exclude values <= vmin and values >= vmax, if they are specified.

Returns Copy of df with stats appended as new columns.

Return type DataFrame

craterpy.stats.**ejecta_stats**(*df*, *cds*, *ejrad=2*, *buffer=1*, *stats=None*, *plot=None*, *save=None*, *prefix=None*, *vmin=- inf*, *vmax=inf*, *strict=False*, *verbosity=1*)

Compute the specified stats from acestats.py on a circular ejecta ROI extending ejsize crater radii from the crater center. Return results as DataFrame with stats appended as extra columns.

Parameters

- **df** (*pandas.DataFrame object*) – Contains the crater locations and sizes to locate ROIs in aceDS. Also form the basis of the returned DataFrame
- **cds** (*CraterpyDataset object*) – CraterpyDataset of image data used to compute stats.
- **ejrad** (*int, float*) – The radius of ejecta blanket measured in crater radii from the crater center to the ejecta extent.
- **buffer** (*int, float*) – Extra buffer around crater rim to mask (multiplicative factor of crater radius). Defaults to 1 (no extra buffer).
- **stats** (*Array-like of str*) – Indicates the stat names to compute. Stat functions must be defined in acestats.py. Default: all stats in acestats.py.
- **plot** (*bool*) – Plot the ejecta ROI.
- **vmin** (*int, float*) – The minimum valid image pixel value. Filter all values lower.
- **vmax** (*int, float*) – The maximum valid image pixel value. Filter all values higher.
- **strict** (*bool*) – How strict the (vmin, vmax) range is. If true, exclude values <= vmin and values >= vmax, if they are specified.

Returns Copy of df with stats appended as new columns.

Return type DataFrame

craterpy.stats.**histogram**(*roi*, *bins*, *hmin=None*, *hmax=None*, *skew=False*, *verbose=False*, ***kwargs*)

Return histogram, bins of histogram computed on ROI. See np.histogram for full usage and optional parameters. Set verbose=True to print a summary of the statistics.

7.1.4 Craterpy masking module

Mask CraterRoi objects.

craterpy.masking.**circle_mask**(*shape*, *radius*, *center=None*)

Return boolean array of True inside circle of radius at center.

Parameters

- **shape** (*tuple of int*) – Shape of output boolean mask array of the form (ysize, xsize).
- **radius** (*int or float*) – Radius of circle [pixels].
- **center** (*tuple of int*) – Two element tuple of (yindex, xindex) of circle center (defaults to center of the mask).

Returns **mask**

Return type numpy 2D array

Examples

```
>>> circle_mask((3,3), 1)
array([[False,  True, False],
       [ True,  True,  True],
       [False,  True, False]], dtype=bool)
>>> circle_mask((3,3), 1, center=(1, 2))
array([[False, False,  True],
       [False,  True,  True],
       [False, False,  True]], dtype=bool)
```

craterpy.masking.**ellipse_mask**(*shape*, *ysize*, *xsize*, *center=None*)

Return numpy 2D boolean array of True inside ellipse.

Parameters

- **shape** (*tuple of int*) – Shape of output boolean mask array of the form (ysize, xsize).
- **ysize** (*int or float*) – Vertical semi-axis [pixels].
- **xsize** (*int or float*) – Horizontal semi-axis [pixels].
- **center** (*tuple of int*) – Two element tuple of (yindex, xindex) of ellipse center (defautls to center of the mask).

Returns mask

Return type numpy 2D array

Examples

```
>>> ellipse_mask((4,5), 1.5, 3)
array([[False, False,  True, False, False],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [False, False,  True, False, False]], dtype=bool)
>>> ellipse_mask((4,5), 1.5, 3, center=(1.5, 3))
array([[False, False, False,  True, False],
       [False,  True,  True,  True,  True],
       [False,  True,  True,  True,  True],
       [False, False, False,  True, False]], dtype=bool)
```

craterpy.masking.**ring_mask**(*shape*, *rmin*, *rmax*, *center=None*)

Return bool array of True in a circular ring from rmin to rmax.

Parameters

- **shape** (*tuple of int*) – Shape of output boolean mask array of the form (ysize, xsize).
- **rmin** (*int or float*) – Inner ring radius [pixels].
- **rmax** (*int or float*) – Outer ring radius [pixels].
- **center** (*tuple of int*) – Two element tuple of (yindex, xindex) of ellipse center (defautls to center of the mask).

Returns mask

Return type numpy 2D array

Examples

```
>>> ring_mask((5,5), 1, 2)
array([[False, False,  True, False, False],
       [False,  True, False,  True, False],
       [ True, False, False, False,  True],
       [False,  True, False,  True, False],
       [False, False,  True, False, False]], dtype=bool)
>>> ring_mask((5,5), 0.5, 1.5)
array([[False, False, False, False, False],
       [False,  True,  True,  True, False],
       [False,  True, False,  True, False],
       [False,  True,  True,  True, False],
       [False, False, False, False, False]], dtype=bool)
```

craterpy.masking.**crater_floor_mask**(*croi*, *buffer=1*)

Return bool mask of interior of a crater in a CraterRoi.

Parameters

- **croi** (CraterRoi) – Crater region of interest specifying crater to mask
- **buffer** (*int or float*) – Extra buffer around crater rim to mask (multiplicative factor of crad)

Returns mask

Return type numpy 2D array

Examples

craterpy.masking.**crater_ring_mask**(*croi*, *rmin*, *rmax*)

Return bool mask of interior of a ring aroun a crater in a CraterRoi.

Ring is calculated by subtracting an inner ellipse mask from an outer ellipse mask which are specified by rmin and rmax, respectively.

Parameters

- **croi** (CraterRoi) – Crater region of interest specifying crater to mask
- **rmin** (*int or float*) – Inner ring radius [km].
- **rmax** (*int or float*) – Outer ring radius [km].

Returns mask

Return type numpy 2D array

Examples

craterpy.masking.**polygon_mask**(*croi*, *poly_verts*)

Mask the region inside a polygon given by poly_verts.

Uses the matplotlib.Path module and included contains_points() method to calculate the interior of a polygon specified as a list of (lat, lon) vertices.

Parameters

- **croi** (CraterRoi) – Crater region of interest being masked.

- **poly_verts** (*list of tuple*) – List of (lon, lat) polygon vertices.

Returns mask

Return type numpy 2D array

Example

```
>>> import os.path as p
>>> f = p.join(p.dirname(p.abspath('__file__')), 'examples', 'moon.tif')
>>> cds = CraterpyDataset(f, radius=1737)
>>> croi = CraterRoi(cds, -27.2, 80.9, 207)  # Humboldt crater
>>> poly = [[-27.5, 80.5], [-28, 80.5], [-28, 81], [-27.5, 81]]
>>> mask = polygon_mask(cds, roi, extent, poly)
>>> croi.mask(mask)
>>> croi.plot()
```

7.1.5 Craterpy plotting module

This file contains helper functions for plotting.

craterpy.plotting.**plot_CraterRoi**(*croi*, *figsize=(4, 4)*, *title=None*, *cmap='gray'*, ***kwargs*)

Plot 2D CraterRoi.

The plot offers limited arguments for basic customization. It is further customizable by supplying valid matplotlib.imshow() keyword-arguments. See matplotlib.imshow for full documentation.

Parameters

- **roi** (*CraterRoi object*) – 2D CraterRoi to plot.
- **figsize** (*tuple*) – Length and width of plot in inches (default 4in x 4in).
- **title** (*str*) – Plot title.
- **cmap** (*str*) – Color map to plot (default 'gray'). See matplotlib.cm for full list.

Other Parameters ****kwargs** (*object*) – Keyword arguments to pass to imshow. See matplotlib.pyplot.imshow

7.1.6 Craterpy helper functions

PYTHON MODULE INDEX

C

craterpy.masking, 22
craterpy.plotting, 25
craterpy.stats, 21

INDEX

C

calc_mpp() (*craterpy.dataset.CraterpyDataset method*), 17
cds (*craterpy.roi.CraterRoi attribute*), 19
circle_mask() (*in module craterpy.masking*), 22
compute_stats() (*in module craterpy.stats*), 21
crater_floor_mask() (*in module craterpy.masking*), 24
crater_ring_mask() (*in module craterpy.masking*), 24
crater_stats() (*in module craterpy.stats*), 21
craterpy.masking
    module, 22
craterpy.plotting
    module, 25
craterpy.stats
    module, 21
CraterpyDataset (*class in craterpy.dataset*), 17
CraterRoi (*class in craterpy.roi*), 19

E

ejecta_profile_stats() (*in module craterpy.stats*), 21
ejecta_stats() (*in module craterpy.stats*), 21
ellipse_mask() (*in module craterpy.masking*), 23
extent (*craterpy.roi.CraterRoi attribute*), 20

F

filter() (*craterpy.roi.CraterRoi method*), 20

G

get_roi() (*craterpy.dataset.CraterpyDataset method*), 19

H

histogram() (*in module craterpy.stats*), 22

I

inbounds() (*craterpy.dataset.CraterpyDataset method*), 18
is_global() (*craterpy.dataset.CraterpyDataset method*), 18

M

mask() (*craterpy.roi.CraterRoi method*), 20
module
    craterpy.masking, 22
    craterpy.plotting, 25
    craterpy.stats, 21

P

plot() (*craterpy.roi.CraterRoi method*), 20
plot_CraterRoi() (*in module craterpy.plotting*), 25
polygon_mask() (*in module craterpy.masking*), 24
ppd (*craterpy.dataset.CraterpyDataset attribute*), 17

R

radius (*craterpy.dataset.CraterpyDataset attribute*), 17
radius (*craterpy.roi.CraterRoi attribute*), 19
ring_mask() (*in module craterpy.masking*), 23
roi (*craterpy.roi.CraterRoi attribute*), 20

S

save() (*craterpy.roi.CraterRoi method*), 20

W

wsize (*craterpy.roi.CraterRoi attribute*), 19