focal_stats

Release 0.0.1

Jasper Roebroek

Aug 11, 2022

GETTING STARTED

This module aims to provide focal statistics for python, that runs without the installation of extensive GIS packages. The only dependency is numpy. Additionally, the package provides sliding window functionality to implement your own focal statistics functions (see Tutorials). This is implemented in line with `numpy.lib.stride_tricks.sliding_window_view()`, but extends its functionality.

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

1.1.1 Required dependencies

- numpy

1.1.2 Instructions

Install with

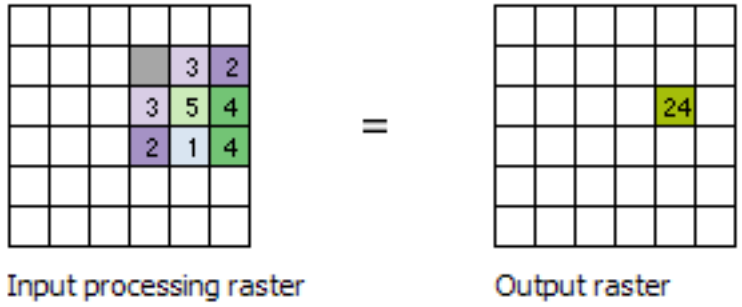
```
conda install --channel conda-forge focal-stats
```

1.2 Focal statistics

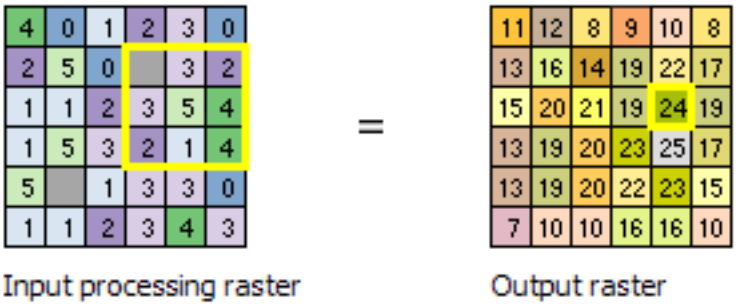
Focal statistics are functions calculated on the neighborhood of the data, often referred to as sliding, or rolling window operations on 2D data. Functions are statistical and range from mean and standard deviation to quantiles and mode (majority).

Conceptually, the algorithm iterates over each pixel of a raster, and looks in all four directions to calculate the focal statistic. The neighborhoods can overlap, but might also be stacked next to each other, so a sparse output is generated.

An example from the ArcGIS documentation, considering the focal sum of a raster with a specified neighborhood of 3x3 pixels. The values in this window are summed and placed in the output array at the location of the most central pixel in the window:



The second example shows what the same example looks like when a complete raster is considered:



The statistical operations implemented in this package are:

- mean: *focal_mean()*
- minimum: *focal_min()*
- maximum: *focal_max()*
- standard deviation: *focal_std()*
- summation: *focal_sum()*
- majority/mode: *focal_majority()*
- correlation: *focal_correlation()*

None

Note: This tutorial was generated from an IPython notebook that can be downloaded here.

1.3 Usage example

```
import focal_stats
import rasterio as rio
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
Matplotlib is building the font cache; this may take a moment.
```

```
os.chdir("../../../")
```

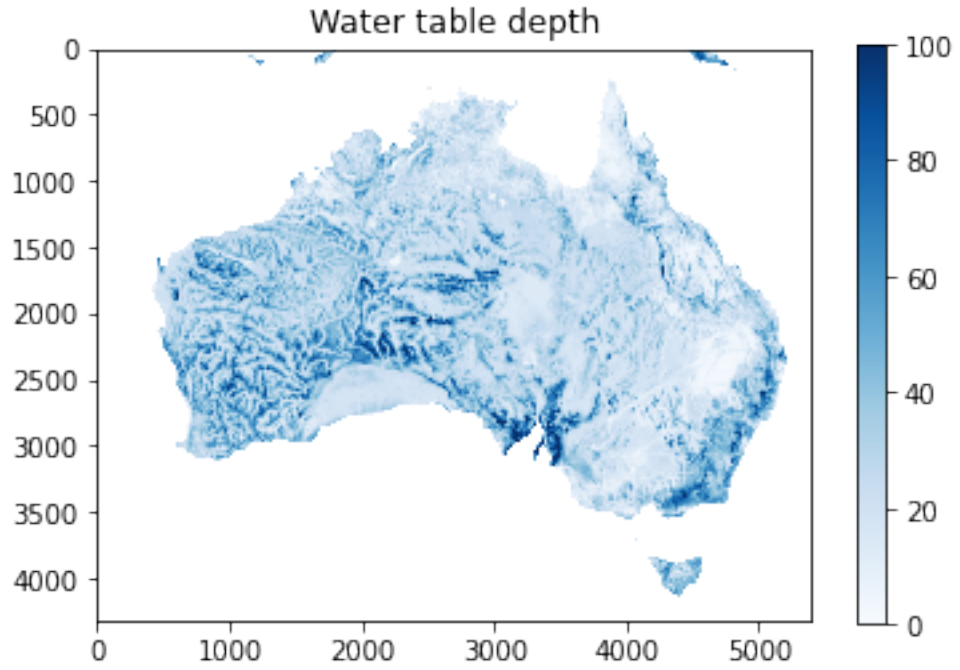
Loading raster (containing water table depth (Fan et al., 2017)).

```
with rio.open("data/wtd.tif") as f:
    a = f.read(1).astype(np.float64)
    a[a == -999.9] = np.nan
```

Inspecting the data

```
plt.imshow(a, cmap='Blues', vmax=100)
plt.title("Water table depth")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7ff3c4aec550>
```

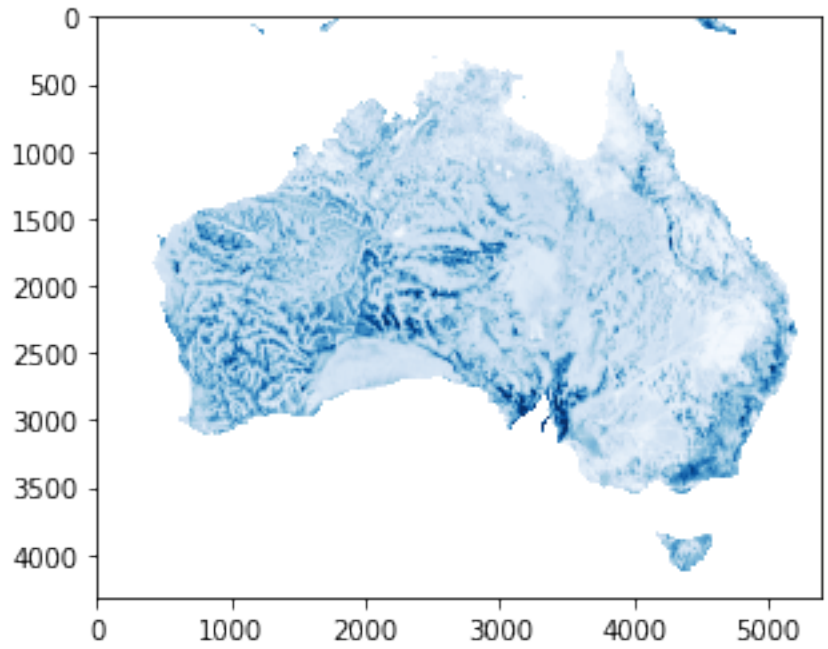


1.3.1 Focal statistics

Calculation of the focal mean:

```
plt.imshow(focal_stats.focal_mean(a, window_size=15), vmax=100, cmap="Blues")
```

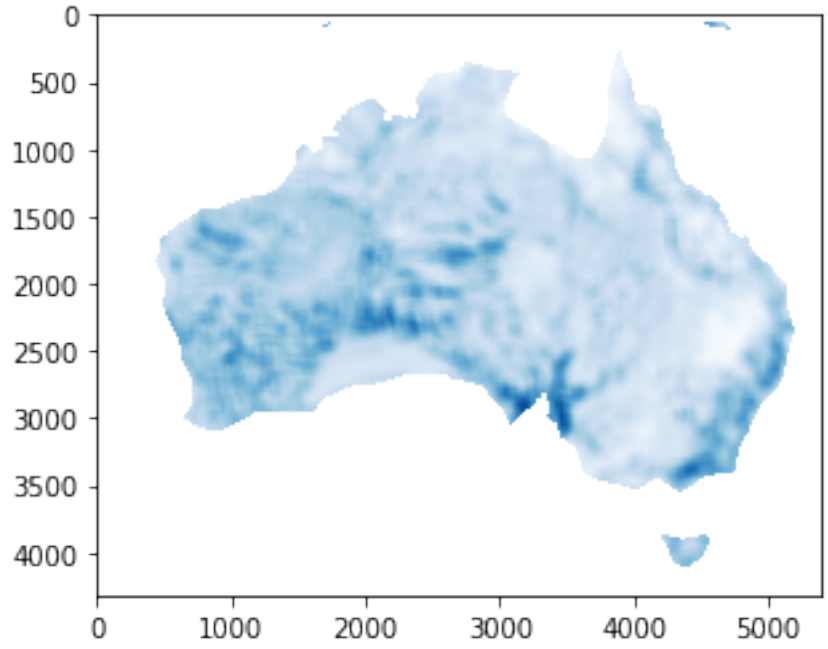
```
<matplotlib.image.AxesImage at 0x7ff3c4184b50>
```



This looks quite similar to the input raster, but with smoothing applied. Let's try a higher window_size, which should increase the smoothing

```
plt.imshow(focal_stats.focal_mean(a, window_size=101), vmax=100, cmap="Blues")
```

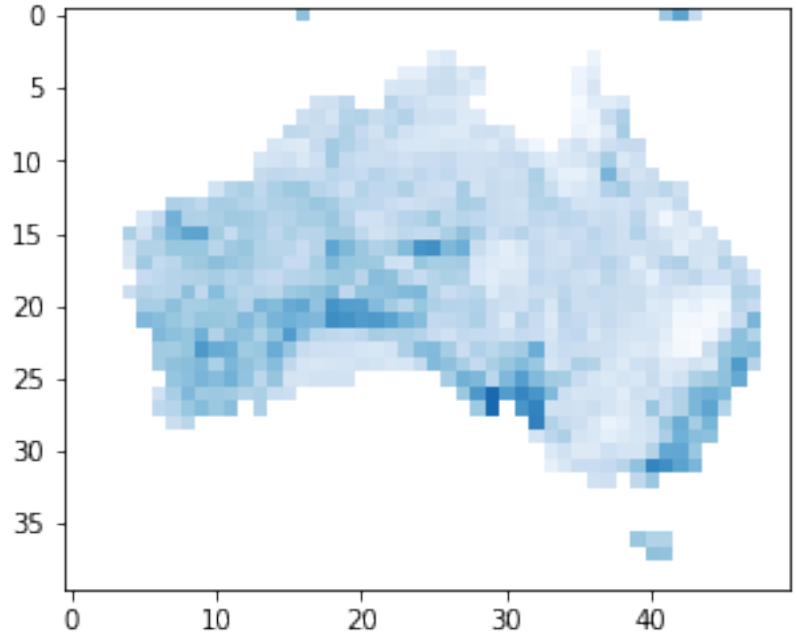
```
<matplotlib.image.AxesImage at 0x7ff3c4112c90>
```



This same functionality can be used to reduce the shape of the raster based on this window_size.

```
x = focal_stats.focal_mean(a, window_size=108, reduce=True)
plt.imshow(x, vmax=100, cmap="Blues")
```

```
<matplotlib.image.AxesImage at 0x7ff3c4084b90>
```



The shape of this new raster is exactly 108 times smaller than the input raster. Note that for this to work both x and y-axes need to be divisible by the window size.

None

Note: This tutorial was generated from an IPython notebook that can be downloaded here.

1.4 Creating custom focal statistic function

```
import focal_stats
import rasterio as rio
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
os.chdir("../../../")
```

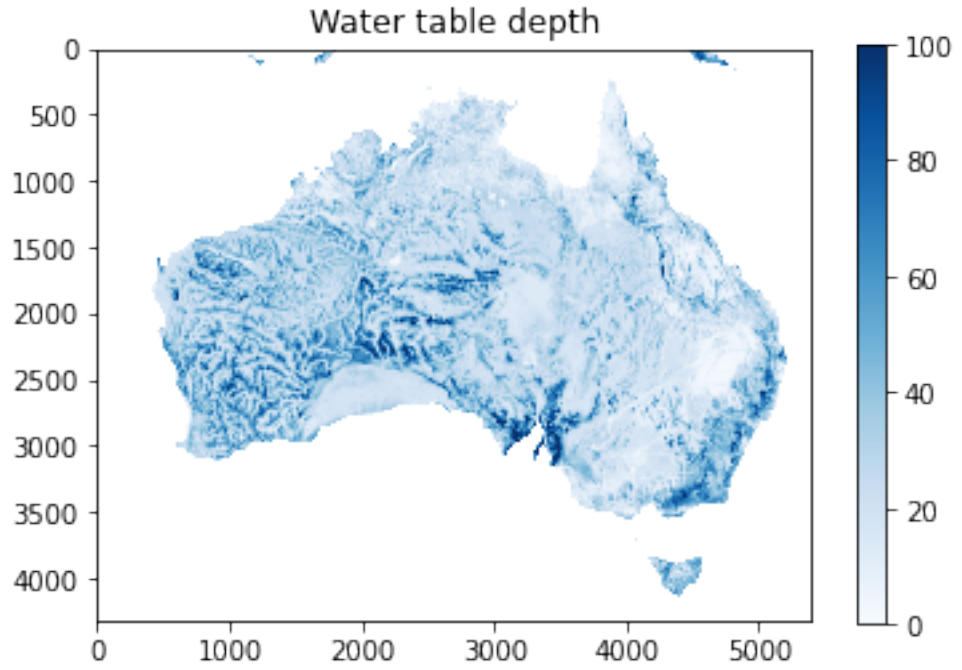
Loading raster (containing water table depth (Fan et al., 2017)).

```
with rio.open("data/wtd.tif") as f:
    a = f.read(1).astype(np.float64)
    a[a == -999.9] = np.nan
```

Inspecting the data

```
plt.imshow(a, cmap='Blues', vmax=100)
plt.title("Water table depth")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f3f63ec3e90>
```



1.4.1 Creating custom focal mean

Firstly, a windowed version of the input raster needs to be defined.

```
a_windowed = focal_stats.rolling_window(a, window_size=5)
print(a.shape, a_windowed.shape)
```

```
(4320, 5400) (4316, 5396, 5, 5)
```

This windowed version has a slightly different shape on the first two axes. This is because there is no window behaviour defined on the edges. If this is undesired the original array can be padded with the missing number of columns and rows with numpy.pad. Through this function many different edge value assumptions can be made. Here I use the example of continuing with the closest values.

```
a_padded = np.pad(a, pad_width=2, mode='edge')
a_windowed_padded = focal_stats.rolling_window(a_padded, window_size=5)
print(a.shape, a_windowed_padded.shape)
```

```
(4320, 5400) (4320, 5400, 5, 5)
```

This has the result that the input and output raster share their first two axes.

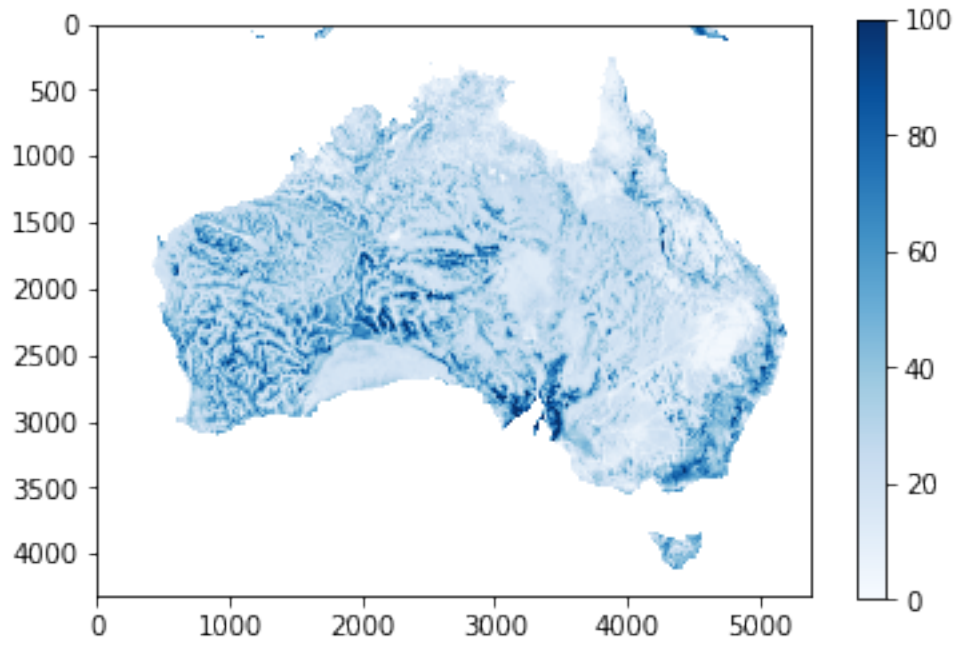
Now the only thing that needs to happen is a mean operation on the third and fourth axes:

```
a_mean = a_windowed.mean(axis=(2, 3))
```

Plotting this shows that the operation generates an image that is very close to the original raster, with some limited smoothing

```
plt.imshow(a_mean, cmap="Blues", vmax=100)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f3f62503510>
```



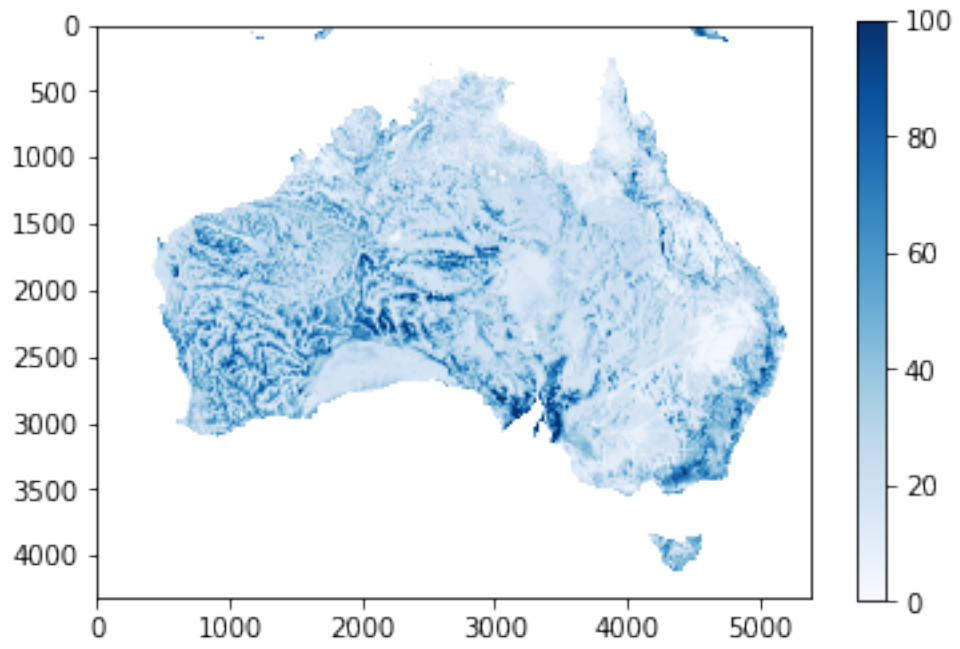
This can be captured in a custom focal_mean function as follows:

```
def focal_mean(a, window_size):
    a_windowed = focal_stats.rolling_window(a, window_size=window_size)
    return a_windowed.mean(axis=(2, 3))
```

Resulting in the same image:

```
plt.imshow(focal_mean(a, window_size=5), cmap="Blues", vmax=100)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f3f623f4550>
```



Note that if a single NaN-value was present in the window, it results in a NaN-value. I dealt with this by inserting 0 in the pixels with NaN-values and using the sum of this array divided by the number of valid values per window (e.g. `rolling_sum(~np.isnan(a), window_size=5)`).

1.5 What's new

1.6 API reference

1.6.1 Focal statistics

Focal statistics functions operating on array-like input data. They share functionality of `window_size` (determining the size of the sliding window), `mask` (superseding `window_size` and allowing for non-rectangular windows), `fraction_accepted` (nan-behaviour based on the fraction of available data in the input window) and `reduce` (switch between returning same shape as input data and patching the sliding window without overlapping, leading to a much smaller output array). *`focal_correlation()`* calculates the correlation between two arrays in contrast to the other functions that operate on a single array.

`focal_min`	Focal minimum
`focal_max`	Focal maximum
`focal_mean`	Focal mean
`focal_std`	Focal standard deviation
`focal_sum`	Focal summation
`focal_majority`	Focal majority
`focal_correlation`	Focal correlation

focal_stats.focal_min

focal_stats.**focal_min**()

Focal minimum

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_max

focal_stats.**focal_max**()

Focal maximum

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_mean

focal_stats.**focal_mean**()

Focal mean

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_std

focal_stats.**focal_std**()

Focal standard deviation

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.
- **std_df** (*{1,0}, optional*) – Degrees of freedom; meaning if the function is divided by the count of observations or the count of observations minus one. Should be 0 or 1. See numpy.std() for documentation.

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_sum

focal_stats.**focal_sum**()

Focal summation

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_majority

focal_stats.**focal_majority**()

Focal majority

Parameters

- **a** (*array_like*) – Input array (2D)
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as (window_size, window_size). If a list is provided it needs to have the same number of entries as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array (2D). If provided, its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of valid cells (not NaN) per window that is deemed acceptable 0: all window are calculated if at least 1 value is present 1: only windows completely filled with values are calculated 0-1: fraction of acceptability
- **verbose** (*bool, optional*) – Verbosity with timing. False by default
- **reduce** (*bool, optional*) – The way in which the windowed array is created. If true, all values are used exactly once. If False (which is the default), values are reduced and the output array has the same shape as the input array, albeit with a border of nans where there are not enough values to calculate the cells.
- **majority_mode** (*{"nan", "ascending", "descending"}, optional*) – Differt modes of dealing with more than one value occuring equally often:
  - nan: when more than one class has the same score NaN will be assigned
  - ascending: the first occurrence of the maximum count will be assigned
  - descending: the last occurrence of the maximum count will be assigned

Returns

if reduce is False:

numpy ndarray of the function applied to input array a. The shape will be the same as the input array. The border of the map will be filled with nan, because of the lack of data to calculate the border. In the future other behaviours might be implemented. To obtain only the useful cells the following might be done:

```
>>> window_size = 5
>>> fringe = window_size // 2
>>> ind_inner = np.s_[fringe:-fringe, fringe:-fringe]
>>> a = a[ind_inner]
```

in which case a will only contain the cells for which all data was present

if reduce is True:

numpy ndarray of the function applied on input array a. The shape will be the original shape divided by the window_size. Dimensions remain equal. No border of NaN values is present.

Return type

ndarray

focal_stats.focal_correlation

focal_stats.**focal_correlation**()

Focal correlation

Parameters

- **a** (*array-like*) – Input arrays that will be correlated. If not present in dtype `float64` it will be converted internally. They have exatly the same shape and have two dimensions.
- **b** (*array-like*) – Input arrays that will be correlated. If not present in dtype `float64` it will be converted internally. They have exatly the same shape and have two dimensions.
- **window_size** (*int, optional*) – Size of the window used for the correlation calculations. It should be bigger than 1, the default is 5.
- **mask** (*array_like, optional*) – A boolean array. Its shape will be used as window_size, and its entries are used to mask every window.
- **fraction_accepted** (*float, optional*) – Fraction of the window that has to contain not-nans for the function to calculate the correlation. The default is 0.7.
- **reduce** (*bool, optional*) – Reuse all cells exactly once by setting a stepsize of the same size as window_size. The resulting raster will have the shape: `shape/window_size`
- **verbose** (*bool, optional*) – Times the correlation calculations

Returns

numpy array of the local correlation. If `reduce` is set to False, the output has the same shape as the input raster, while if `reduce` is True, the output is reduced by the window size: `shape // window_size`.

Return type

ndarray

1.6.2 Rolling functions

This module additionally implements standalone rolling functions, accepting ND arrays and accepting `window_size`, `mask` and `reduce` similarly to the focal statistics functionality. It does however not guard against NaN occurrences specifically, staying with the raw numpy behaviour. The functions are intended to be used to define custom focal statistics functionality, potentially in higher than 2D dimensionality.

rolling_window(a, *[, window_size, mask, ...])	Takes an array and returns a windowed version
rolling_sum(a, *[, window_size, mask, reduce])	Takes an array and returns the rolling sum.
rolling_mean(a, *[, window_size, mask, reduce])	Takes an array and returns the rolling mean.

focal_stats.rolling_window

focal_stats.**rolling_window**(*a*, *, *window_size=None*, *mask=None*, *flatten=False*, *reduce=False*, ***kwargs*)

Takes an array and returns a windowed version

Parameters

- **a** (*array_like*) – Input array
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over a. Should be a positive integer. If it is an integer it will be interpreted as `(window_size, )`

`* a.ndim`. If a list is provided it needs to have the same length as the number of dimensions as a.

- **mask** (*array_like, optional*) – A boolean array. Its shape will be used as `window_size`, and its entries are used to mask every window, resulting in dimensionality `a.ndim + 1` as the final result, just as in the case of *flatten* is True.
- **flatten** (*bool, optional*) – Flag to flatten the windowed view to 1 dimension. The shape of the returned array if set to True will be:

  ***reduce* == False:**
  shape : (s - window_size + 1) + (np.prod(window_size),)

  ***reduce* == True:**
  shape : (s // window_size) + (np.prod(window_size),)

  If set to False (which is the default) the shape of the window will not change and the data will be added in as many dimensions as the input array. The shape will be:

  ***reduce* == False:**
  shape : (s - window_size + 1) + (window_size)

  ***reduce* == True:**
  shape : (s // window_size) + (window_size)

  False has the nice property of returning a view, not copying the data while if True is passed, all the data will be copied. This can be very slow and memory intensive for large arrays.
- **reduce** (*bool, optional*) – Reuse data if set to False (which is the default) in which case an array will be returned with dimensions that are close to the original; see `flatten`. If set to true, every entry is used exactly once. Creating much smaller dimensions.
- **kwargs** (*dict, optional*) – Arguments for `as_strided()`, notably `subok` and `writeable` (see numpy documentation).

Returns
windowed array of the data

Return type
ndarray

Raises
ValueError –

- window_size too bigger than on of the dimensions of the input array
- if reduce is True, the window_size needs to be an exact divisor for all dimensions of the input array

focal_stats.rolling_sum

`focal_stats.`**rolling_sum**(*a*, ***, *window_size=None*, *mask=None*, *reduce=False*)

Takes an array and returns the rolling sum. Not suitable for arrays with NaN values.

Parameters

- **a** (*array_like*) – Input array
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over `a`. Should be a positive integer. If it is an integer it will be interpreted as `(window_size, ) * a.ndim`. If a list is provided it needs to have the same length as the number of dimensions as a.

- **mask** (*array_like, optional*) – A boolean array. Its shape will be used as `window_size`, and its entries are used to mask every window, resulting in dimensionality `a.ndim + 1` as the final result, just as in the case of *flatten* is True.
- **reduce** (*bool, optional*) – Reuse data if set to False (which is the default) in which case an array will be returned with dimensions that are close to the original; see `flatten`. If set to true, every entry is used exactly once. Creating much smaller dimensions.

Returns

Rolling sum over array `a`. Resulting shape depends on reduce parameter. See *rolling_window()* for documentation. If a mask is provided, the last dimension has the length of the sum of `mask`.

Return type

ndarray

focal_stats.rolling_mean

`focal_stats.`**rolling_mean**(*a*, *, *window_size=None*, *mask=None*, *reduce=False*)

Takes an array and returns the rolling mean. Not suitable for arrays with NaN values.

Parameters

- **a** (*array_like*) – Input array
- **window_size** (*int, array_like, optional*) – Size of the window that is applied over `a`. Should be a positive integer. If it is an integer it will be interpreted as `(window_size, ) * a.ndim`. If a list is provided it needs to have the same length as the number of dimensions as a.
- **mask** (*array_like, optional*) – A boolean array. Its shape will be used as `window_size`, and its entries are used to mask every window, resulting in dimensionality `a.ndim + 1` as the final result, just as in the case of *flatten* is True.
- **reduce** (*bool, optional*) – Reuse data if set to False (which is the default) in which case an array will be returned with dimensions that are close to the original; see `flatten`. If set to true, every entry is used exactly once. Creating much smaller dimensions.

Returns

Rolling mean over array `a`. Resulting shape depends on reduce parameter. See *rolling_window()* for documentation. If a mask is provided, the last dimension has the length of the sum of `mask`.

Return type

ndarray

1.7 License

focal_stats is published under a MIT license.

1.8 Indices and tables

- genindex
- modindex
- search

INDEX

F

R