rio-terrain Documentation

Release 0.0.29

Michael Rahnis

Mar 06, 2022

Contents

CHAPTER 1

rio-terrain

Rio-terrain provides a set of rasterio CLI plugins to perform common raster operations, and can write slope, aspect and curvature rasters.

1.1 Installation

To install from the Python Package Index:

```
$pip install rio-terrain
```

To install from Anaconda Cloud:

If you are starting from scratch the first thing to do is install the Anaconda Python distribution, add the necessary channels to obtain the dependencies and install rio-terrain.

```
$conda config --append channels conda-forge
$conda install rio-terrain -c mrahnis
```

To install from the source distribution execute the setup script in the rio-terrain directory:

```
$python setup.py install
```

1.2 Examples

TODO

1.3 License

BSD

1.4 Documentation

Latest html

CHAPTER 2

Command Line User Guide

The rasterio command line interface plugins allow you to execute commands that operate on a raster dataset. Online help lists the avalable subcommands, including those added by rio-terrain.

```
$ rio --help
Usage: rio [OPTIONS] COMMAND [ARGS]...

  Rasterio command line interface.

Options:
  -v, --verbose           Increase verbosity.
  -q, --quiet             Decrease verbosity.
  --aws-profile TEXT      Select a profile from the AWS credentials file
  --aws-no-sign-requests  Make requests anonymously
  --aws-requester-pays    Requester pays data transfer costs
  --version               Show the version and exit.
  --gdal-version
  --help                  Show this message and exit.

Commands:
  aspect       Calculates aspect of a height raster.
  blocks       Write dataset blocks as GeoJSON features.
  bounds       Write bounding boxes to stdout as GeoJSON.
  calc         Raster data calculator.
  clip         Clip a raster to given bounds.
  ...
```

The list below describes the purpose of the individual rio-terrain subcommands. Command usage can be had by accessing the `--help` of each command.

2.1 aspect

```
$ rio aspect --help
Usage: rio aspect [OPTIONS] INPUT OUTPUT

  Calculate aspect of a raster.

  INPUT should be a single-band raster.

  Example:
  rio aspect elevation.tif aspect.tif --pcs compass

Options:
  --neighbors [4|8]          Specifies the number of neighboring cells to use.
  --pcs [compass|cartesian]  Specifies the polar coordinate system.
  -j, --njobs INTEGER        Number of concurrent jobs to run
  -v, --verbose              Enables verbose mode.
  --version                  Show the version and exit.
  --help                     Show this message and exit.
```

2.2 curvature

```
$ rio curvature --help
Usage: rio curvature [OPTIONS] INPUT OUTPUT

  Calculate curvature of a raster.

  INPUT should be a single-band raster.

  Example:
  rio curvature elevation.tif curvature.tif

Options:
  --neighbors [4|8]     Specifies the number of neighboring cells to use.
  --stats / --no-stats  Print basic curvature statistics.
  -j, --njobs INTEGER   Number of concurrent jobs to run
  -v, --verbose         Enables verbose mode.
  --version             Show the version and exit.
  --help                Show this message and exit.
```

2.3 difference

```
$ rio difference --help
Usage: rio difference [OPTIONS] INPUT_T0 INPUT_T1 OUTPUT

  Subtract INPUT_T0 from INPUT_T1.

  INPUT_T0 should be a single-band raster at time t0.
  INPUT_T1 should be a single-band raster at time t1.

  Example:
```

(continues on next page)

(continued from previous page)

```
  rio diff elevation1.tif elevation2.tif, diff2_1.tif

Options:
  -b, --blocks INTEGER  Multiply TIFF block size by an amount to make chunks
  -j, --njobs INTEGER   Number of concurrent jobs to run.
  -v, --verbose         Enables verbose mode.
  --version             Show the version and exit.
  --help                Show this message and exit.
```

2.4 extract

```
$ rio extract --help
Usage: rio extract [OPTIONS] INPUT CATEGORICAL OUTPUT

  Extract regions from a raster by category.

  INPUT should be a single-band raster.
  CATEGORICAL should be a single-band raster with categories to extract.

  The categorical data may be the input raster or another raster.

  Example:
  rio extract diff.tif categorical.tif extract.tif -c 1 -c 3

Options:
  -c, --category INTEGER  Category to extract.
  -j, --njobs INTEGER     Number of concurrent jobs to run
  -v, --verbose           Enables verbose mode.
  --version               Show the version and exit.
  --help                  Show this message and exit.
```

2.5 label

```
$ rio label --help
Usage: rio label [OPTIONS] INPUT OUTPUT

  Label regions in a raster.

  INPUT should be a single-band raster.

  Example:
  rio label blobs.tif labeled_blobs.tif

Options:
  --diagonals / --no-diagonals  Label diagonals as connected
  --zeros / --no-zeros          Use the raster nodata value or zeros for False
                                condition
  -j, --njobs INTEGER           Number of concurrent jobs to run
  -v, --verbose                 Enables verbose mode.
  --version                     Show the version and exit.
  --help                        Show this message and exit.
```

2.6 mad

```
$ rio mad --help
Usage: rio mad [OPTIONS] INPUT OUTPUT

  Calculate a median absolute deviation raster.

  INPUT should be a single-band raster.

  Example:
  rio mad elevation.tif mad.tif

Options:
  -n, --neighborhood INTEGER  Neighborhood size in cells.
  -b, --blocks INTEGER        Multiply TIFF block size by an amount to make
                              chunks
  -j, --njobs INTEGER         Number of concurrent jobs to run.
  -v, --verbose               Enables verbose mode.
  --version                   Show the version and exit.
  --help                      Show this message and exit.
```

2.7 quantiles

```
$ rio quantiles --help
Usage: rio quantiles [OPTIONS] INPUT

  Calculate and print quantile values.

  INPUT should be a single-band raster.

  Example:
  rio quantiles elevation.tif -q 0.5 -q 0.9

Options:
  -q, --quantile FLOAT        Print quantile value
  -f, --fraction FLOAT        Randomly sample a fraction of data blocks
  --absolute / --no-absolute  Calculate quantiles based on the set of absolute
                              values
  --describe / --no-describe  Print descriptive statistics to the console
  --plot / --no-plot          Display statistics plots
  -j, --jobs INTEGER          Number of concurrent jobs to run
  -v, --verbose               Enables verbose mode
  --version                   Show the version and exit.
  --help                      Show this message and exit.
```

2.8 slice

```
$ rio slice --help
Usage: rio slice [OPTIONS] INPUT OUTPUT

  Extract regions from a raster by a data range.
```

(continues on next page)

(continued from previous page)

```
  INPUT should be a single-band raster.

  Setting the --keep-data option will return the data values.
  The default is to return a raster of ones and zeros.

  Example:
  rio range diff.tif extracted.tif --minumum -2.0 --maximum 2.0

Options:
  --minimum FLOAT               Minimum value to extract.
  --maximum FLOAT               Maximum value to extract.
  --keep-data / --no-keep-data  Return the input data. Default is to return
                                ones.
  --zeros / --no-zeros          Use the raster nodata value or zeros for False
                                condition
  -j, --njobs INTEGER           Number of concurrent jobs to run
  -v, --verbose                 Enables verbose mode.
  --version                     Show the version and exit.
  --help                        Show this message and exit.
```

2.9 slope

```
$ rio slope --help
Usage: rio slope [OPTIONS] INPUT OUTPUT

  Calculate slope of a raster.

  INPUT should be a single-band raster.

  Example:
  rio slope elevation.tif slope.tif

Options:
  --neighbors [4|8]             Specifies the number of neighboring cells to
                                use.
  -u, --units [grade|degrees]   Specifies the units of slope.
  -b, --blocks INTEGER          Multiply TIFF block size by an amount to make
                                chunks
  -j, --njobs INTEGER           Number of concurrent jobs to run.
  -v, --verbose                 Enables verbose mode.
  --version                     Show the version and exit.
  --help                        Show this message and exit.
```

2.10 std

```
$ rio std --help
Usage: rio std [OPTIONS] INPUT OUTPUT

  Calculate a standard-deviation raster.
```

(continues on next page)

(continued from previous page)

```
  INPUT should be a single-band raster.

  Example:
  rio std elevation.tif stddev.tif

Options:
  -n, --neighborhood INTEGER  Neigborhood size in cells.
  -b, --blocks INTEGER        Multiply TIFF block size by an amount to make
                              chunks
  -j, --njobs INTEGER         Number of concurrent jobs to run
  -v, --verbose               Enables verbose mode.
  --version                   Show the version and exit.
  --help                      Show this message and exit.
```

2.11 threshold

```
$ rio threshold --help
Usage: rio threshold [OPTIONS] INPUT UNCERTAINTY OUTPUT LEVEL

  Threshold a raster with an uncertainty raster.

  INPUT should be a single-band raster.
  UNCERTAINTY should be a single-band raster representing uncertainty.

  Example:
  rio threshold diff.tif uncertainty.tif, detected.tif 1.68

Options:
  -j, --njobs INTEGER  Number of concurrent jobs to run.
  -v, --verbose        Enables verbose mode.
  --version            Show the version and exit.
  --help               Show this message and exit.
```

2.12 uncertainty

```
$ rio uncertainty --help
Usage: rio uncertainty [OPTIONS] UNCERTAINTY0 UNCERTAINTY1 OUTPUT

  Calculate a level-of-detection raster.

  UNCERTAINTY0 should be a single-band raster representing level of uncertainty at␣
↪time 0.
  UNCERTAINTY1 should be a single-band raster representing level of uncertainty at␣
↪time 1.

  Example:
  rio uncertainty roughness_t0.tif roughness_t1.tif uncertainty.tif

Options:
  --instrumental0 FLOAT  Instrumental or minimum uncertainty for the first
                         raster.
```

(continues on next page)

(continued from previous page)

```
  --instrumental1 FLOAT  Instrumental or minimum uncertainty for the second
                         raster.
  -j, --njobs INTEGER    Number of concurrent jobs to run.
  -v, --verbose          Enables verbose mode.
  --version              Show the version and exit.
  --help                 Show this message and exit.
```

CHAPTER 3

rio-terrain

3.1 rio_terrain Package

3.1.1 `terrain` Module

rio_terrain.core.terrain.**aspect**(*arr*, *res=(1, 1)*, *pcs='compass'*, *neighbors=4*)
Calculates aspect.

Parameters

- **arr** (*ndarray*) – 2D numpy array
- **res** (*tuple*) – tuple of raster cell width and height
- **north** (*str, optional*) – choice of polar coordinate system

Returns 2D numpy array representing slope aspect

Return type aspect (ndarray)

rio_terrain.core.terrain.**curvature**(*arr*, *res=(1, 1)*, *neighbors=4*)
Calculates curvature.

Parameters

- **arr** (*ndarray*) – 2D numpy array
- **res** (*tuple*) – tuple of raster cell width and height

Returns 2D numpy array representing surface curvature

Return type curvature (ndarray)

rio_terrain.core.terrain.**slope**(*arr*, *res=(1, 1)*, *units='grade'*, *neighbors=4*)
Calculates slope.

Parameters

- **arr** (*ndarray*) – 2D numpy array

- **res** (*tuple*) – tuple of raster cell width and height
- **units** (*str, optional*) – choice of grade or degrees
- **neighbors** (*int, optional*) – use four or eight neighbors in calculation

Returns 2D numpy array representing slope

Return type slope (ndarray)

3.1.2 windowing Module

rio_terrain.core.windowing.**block_count**(*shape*, *block_shapes*, *band=1*)
Determine the number of blocks in a band

Parameters

- **shape** (*tuple*) – tuple containing raster height and width in cells
- **block_shapes** (*tuple*) – block shapes for a rasterio read source
- **band** (*int*) – raster band to count on

Returns number of blocks in the raster

Return type result (int)

rio_terrain.core.windowing.**bounds_window**(*bounds*, *affine*, *constrain=True*)
Create a full cover rasterio-style window

Parameters

- **bounds** (*tuple*) –
- **affine** (*Affine*) –

Returns row_slice (tuple) col_slice (tuple)

rio_terrain.core.windowing.**chunk_dim**(*dim*, *chunk_size*, *min_size=None*)
Chunk a 1D array

rio_terrain.core.windowing.**chunk_dims**(*shape*, *chunk_shape*, *min_size=None*)
Chunk a 2D array

rio_terrain.core.windowing.**expand_window**(*window*, *src_shape*, *margin=10*)
Expand a window by a margin

Parameters

- **window** (*Window*) –
- **src_shape** (*tuple*) –
- **margin** (*int*) –

Returns result (Window)

rio_terrain.core.windowing.**intersect_bounds**(*bbox0*, *bbox1*)
Get the intersection in w s e n

Parameters

- **bbox0** (*tuple*) –
- **bbox1** (*tuple*) –

Returns coordinate bounds (w, s, e, n)

Return type bounds (tuple)

rio_terrain.core.windowing.**is_raster_aligned**(*src0*, *src1*)
Check two rasters for cell alignment

Parameters

- **src0** – rasterio read source
- **src1** – rasterio read source

Returns True if the raster source cells align

Return type result (bool)

rio_terrain.core.windowing.**is_raster_congruent**(*src0*, *src1*, *band=1*)
Tests two rasters for coincident bounds.

Parameters

- **src0** – rasterio read source
- **src1** – rasterio read source

Returns True if the rasters are coincident

Return type result (bool)

rio_terrain.core.windowing.**is_raster_intersecting**(*src0*, *src1*)
Test two rasters for overlap

rio_terrain.core.windowing.**margins**(*window0*, *window1*)
Size of collar between a pair of windows

Here, window0 is a read window and window1 is a write window

rio_terrain.core.windowing.**slices_to_window**(*rows*, *cols*)

rio_terrain.core.windowing.**subsample**(*blocks*, *probability=1.0*)
Subsample an iterable at a given probability

Parameters

- **blocks** (*iterable*) – an iterable of rasterio windows
- **probability** (*float*) – fraction of blocks to sample

Yields *block (window)* – yield a rasterio window if sampled

rio_terrain.core.windowing.**tile_dim**(*dim*, *tile_size*, *min_size=None*)
Chunk a range using a minimum chunk size

rio_terrain.core.windowing.**tile_dims**(*shape*, *tile_shape*, *min_size=None*)
Chunk a 2D array

rio_terrain.core.windowing.**tile_grid**(*ncols*, *nrows*, *blockxsize*, *blockysize*, *col_offset=0*, *row_offset=0*, *overlap=0*)
Return a generator containing read and write windows with a specified dimensions and overlap

mgrid returns not as expected so used broadcast_arrays instead base_rows, base_cols = np.mgrid[0:h:blockysize, 0:w:blockxsize]

Parameters

- **ncols** (*int*) – raster width in columns
- **nrows** (*int*) – raster height in rows
- **blockxsize** (*int*) – block width in rows

- **blockysize** (*int*) – block height in rows
- **col_offset** (*int*) – columns to offset the grid
- **row_offset** (*int*) – rows to offset the grid
- **overlap** (*int*) – overlap between windows

Yields *window (Window)* – tiled windows over a region

rio_terrain.core.windowing.**tile_grid_intersection**(*src0*, *src1*, *blockxsize=None*, *blockysize=None*)

Generate tiled windows for the intersection between two grids.

Given two rasters having different dimensions calculate read-window generators for each and a write-window generator for the intersecion.

Parameters

- **src0** – rasterio read source
- **src1** – rasterio read source
- **blockxsize** (*int*) – write-window width
- **blockysize** (*int*) – write-window height

Returns read windows for src0 src1_blocks : read windows for src1 write_blocks : write windows for the intersection affine (Affine) : write raster Affine ncols (int) : write raster width in columns nrows (int) : write raster height in rows

Return type src0_blocks

rio_terrain.core.windowing.**trim**(*arr*, *margins*)

Trim a 2D array by a set of margins

rio_terrain.core.windowing.**window_bounds**(*window*, *affine*, *offset='center'*)

Create bounds coordinates from a rasterio window

Parameters

- **window** (*Window*) –
- **affine** (*Affine*) –
- **offset** (*str*) –

Returns coordinate bounds (w, s, e, n)

Return type bounds (tuple)

3.1.3 statistics Module

rio_terrain.core.statistics.**mean**(*src*, *windows*, *njobs*)

Calculates the mean of a rasterio source

Parameters

- **src** – rasterio source
- **windows** – iterable of read and write windows
- **njobs** (*integer*) – number of processing jobs

Returns mean value

Return type mean (float)

mean = 140.043719088 ArcGIS = 140.04371922353

rio_terrain.core.statistics.**minmax**(*src*, *windows*, *njobs*)
Calculates the minimum and maximum values in a rasterio source.

Parameters

- **src** – rasterio source
- **windows** – iterable of read and write windows
- **njobs** (*integer*) – number of processing jobs

Returns minimum value src_max (float) : maximum value

Return type src_min (float)

ArcGIS min = 77.278923034668 ArcGIS max = 218.81454467773

rio_terrain.core.statistics.**stddev**(*src*, *mean*, *windows*, *njobs*)
Calculates the standard deviation of a rasterio source

Parameters

- **src** – rasterio source
- **mean** – mean value
- **windows** – iterable of read and write windows
- **njobs** (*integer*) – number of processing jobs

Returns standard deviation

Return type stddev (float)

stddev = 23.5554506735 ArcGIS = 23.555450665488

3.1.4 focalstatistics Module

rio_terrain.core.focalstatistics.**mad**(*arr*, *size=(3, 3)*)
Calculates the median absolute deviation (MAD) for an array

rio_terrain.core.focalstatistics.**std**(*arr*, *size=(3, 3)*)
Calculates the standard deviation for a neighborhood

rio_terrain.core.focalstatistics.**std_ndimage**(*arr*, *size=(3, 3)*)
Calculates the standard deviation for a neighborhood

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

r

Index