Scarplet Documentation

Release 0.1.0

Robert Sare, George Hilley

Apr 09, 2019

Getting Started

`scarplet` is a tool for topographic feature detection and diffusion or degradation dating. It allows users to define template functions based on the curvature of a landform, and fit them to digital elevation data. As a package, it provides

- A scalable template matching algorithm with `match` and `compare` operations
- A `Scarp` template for fault scarp diffusion dating, and templates for detecting river channels or impact craters
- Flexible template classes

CHAPTER 1

Contents

1.1 Installation

`scarplet` is on PyPI and conda-forge. You can install it with

```
conda install scarplet -c conda-forge
```

or, using `pip`,

```
pip install scarplet
```

The main dependencies are:

- NumPy
- Numexpr
- GDAL and Rasterio
- PyFFTW
- SciPy

A conda installation will install the Python GDAL bindings and PyFFTW. For instructions on manually installing LibFFTW and GDAL, see below.

1.1.1 Installing FFTW3 and pyFFTW

The Fast Fourier Transform library FFTW is a requirement of the `pyfftw` module used by this package. On Ubuntu or Debian, it can be installed with the package manager

```
sudo apt-get install libfftw3-3 libfftw3-bin libfftw3-dev
```

On Mac OS X, you can use Homebrew

```
brew install fftw
```

Then pyFFTW can be install via pip

```
pip install pyfftw
```

There are some known issues with pyFFTW on OS X. It may be necessary to export link paths as environment variable prior to calling pip. See their installation instructions for more details

1.1.2 Installing GDAL

GDAL and `python-gdal` are notoriously tricky to install. Hopefully your system has GDAL installed already; if not, you can install using your OS' package manager.

For example, on Ubuntu or Debian,

```
sudo apt-get install gdal libgdal1h gdal-bin
```

Or, on OS X,

```
brew install gdal
```

Then, the Python bindings to GDAL can be installed. Typically this is as simple as

```
pip install gdal
```

but you may find that the compiler can't find the GDAL header files. Usually this will give a an error like `fatal error: cpl_vsi_error.h: No such file or directory`. To get around this, we need to pass the include path to `pip`:

```
pip install gdal --global-option=build_ext --global-option="-I/usr/include/gdal/"
```

or

```
pip install gdal==$(gdal-config --version) --global-option=build_ext --global-option=
↪"-I/usr/include/gdal/"
```

In my case, with GDAL 1.11.2, this is

```
pip install gdal==1.11.2 --global-option=build_ext --global-option="-I/usr/include/
↪gdal/"
```

Once GDAL is installed, you can go ahead and install the package as usual

```
pip install scarplet
```

1.2 Getting started with scarplet

1.2.1 Input data

Currently `scarplet` handles input data in GeoTiff format. Get a copy of your elevation data as a GeoTiff, and you can load it as

```
import scarplet as sl
data = sl.load('mydem.tif')
```

1.2.2 Choosing a template

If you have gaps in your DEM, no data values will automatically be filled. Then you are ready to choose a template and fit it to your data. These are defined as classes in the WindowedTemplate submodule:

Class	Landform	Use
`Scarp`	Fault scarps, topographic steps	Detecting and morphologic dating of scarp-like landforms
`Channel`	Confined channels	Extracting channel orientations, valley relief
`Crater`	Radially symmetric craters	Measuring crater depth and diffusion dating
`Ricker`	Channels, ridges	Extracting ridge and channel orientations

For example, to use a vertical scarp template, you would import the appropiate template and define a scale and the orientation parameters. In this case, +/- 90 degrees from vertical (*y* direction) captures all scarp orientations.

```
import numpy as np
from scarplet.WindowedTemplate import Scarp
params = {'scale': 100,
          'ang_min': -np.pi / 2,
          'ang_max': np.pi / 2
          }
```

Then, `scarplet`'s `match` function will search over all parameters and return the best-fitting height, relative age, and orientation at each DEM pixel.

```
res = sl.match(data, Scarp, **params)
sl.plot_results(data, res)
```

1.2.3 Viewing matching results

All results are returned as *4* x *height* x *width* arrays of height/amplitude, relative age, orientation, and signal-to-noise-ratio. The easiest way to work with these is to unpack the results and manipulate them as NumPy arrays

```
import matplotlib.pyplot as plt
amp, age, angle, snr = res

fig, ax = plt.subplots(2, 1)
ax[0].hist(np.log10(age.reshape((-1,))), bins=10)
ax[0].set_xlabel('Morphologic age [m$^2$]')

ax[1].hist(angle.reshape((-1,)) * 180 / np.pi, nbins=19)
ax[1].set_xlabel('Orientation [deg.]')
```

1.3 Finding fault scarps

This uses the Scarp template to detect scarp-like landforms and estimate their height and relative age.

It is available as a Jupyter notebook (link) in the repository. Sample data is provided in the data folder.

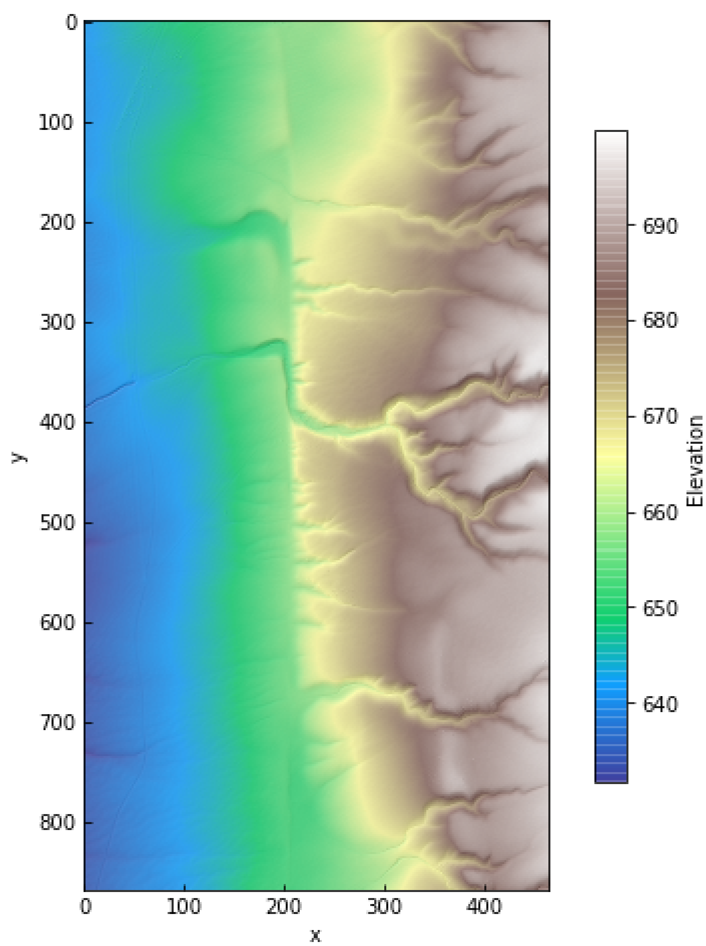
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
np.warnings.filterwarnings('ignore')
```

[2]:
```
import scarplet as sl
from scarplet.datasets import load_carrizo
from scarplet.WindowedTemplate import Scarp
```

The test data comes from the Carrizo Plain section of the San Andreas Fault. It covers part of the Wallace Creek site, a set of offset channels and related scarps and gulleys that have been studied in detail by earthquake geologists and geophysicists (*e.g.*, Sieh and Jahns, 1984; Arrowsmith, et al., 1998).

This high resolution lidar dataset (0.5 m) was downloaded from OpenTopography, a data facility for high-resolution topographic data.

[3]:
```
data = load_carrizo()
dx = data._georef_info.dx
data.plot(color=True, figsize=(8,8))
```



[4]:
```
# Look for scarps of a single morphologic age
params = {'scale': 100,
          'age': 100.,
          'ang_min': -10 * np.pi / 2,
```

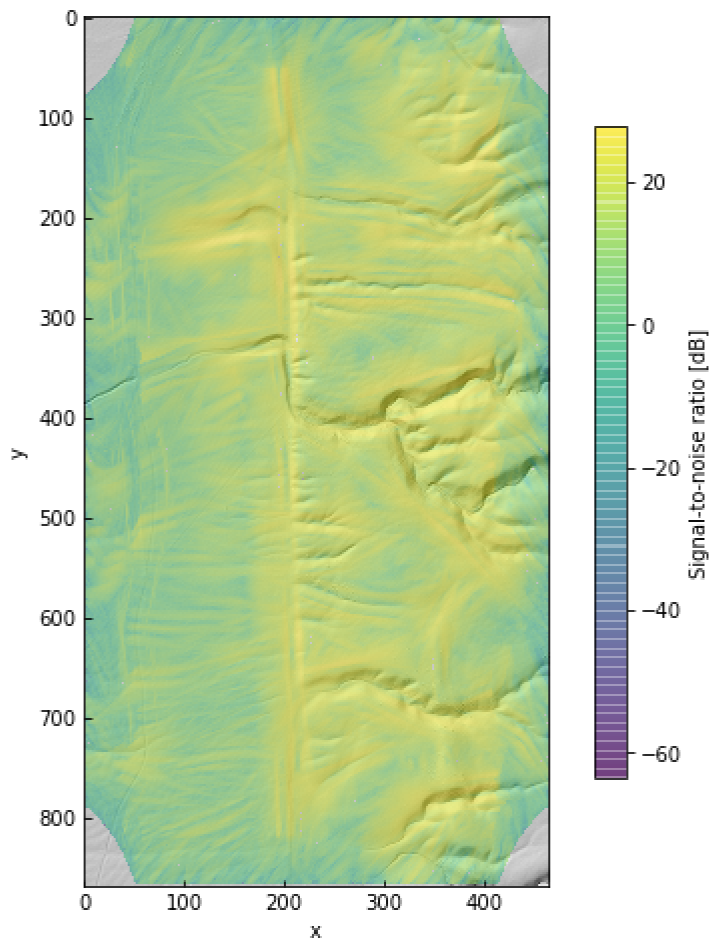
(continues on next page)

(continued from previous page)

```
            'ang_max': 10 * np.pi / 2
           }

res = sl.match(data, Scarp, **params)
```

[5]:
```
amp, age, angle, snr = res
```

[6]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(10 * np.log10(snr), alpha=0.5, cmap='viridis')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Signal-to-noise ratio [dB]')
```



In fact, the sign of the template amplitude is determined by the aspect of the scarp. We will mask by SNR and discard the sign of the amplitude – we just want to see how tall the scarps might be.

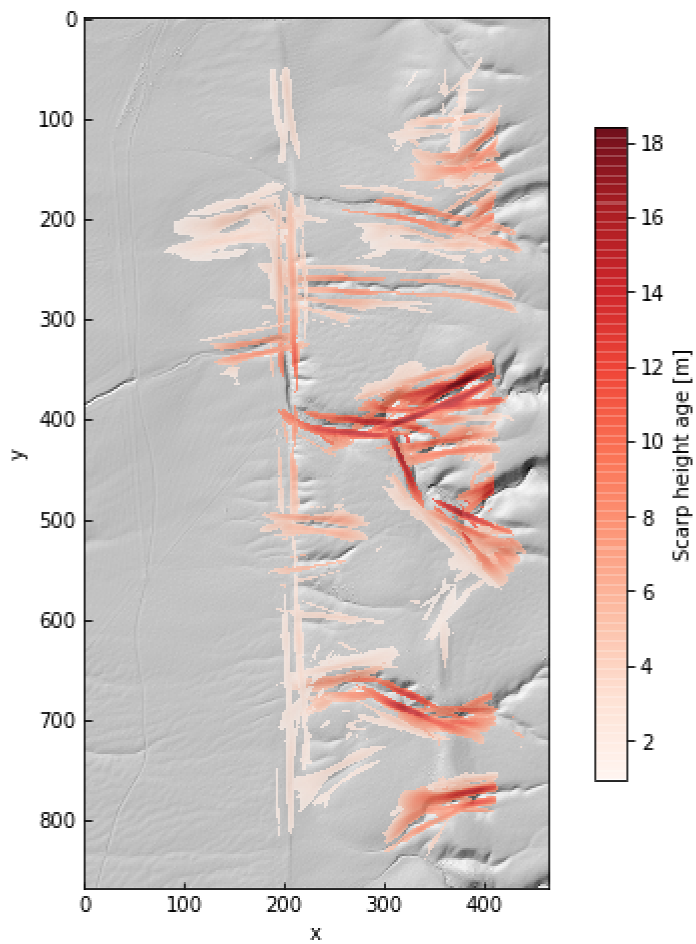
[7]:
```
mask = snr < 100
amp[mask] = np.nan
amp = np.abs(amp)
```

[8]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(amp, alpha=0.75, cmap='Reds')
```

(continues on next page)

(continued from previous page)

```
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Scarp height [m]')
```



From the amplitude field, we can see that there are some false positives – the channels on the right side of the image – as well as amplitude gradients along the main fault trace.

Note that the units of relative age, also called morphologic age, are length^2. This parameter is κt, the product of elapsed time and a diffisivity constant (κ, with units of length^2 time^{-1}). It can be thought of as an estimate of cross-sectional area degraded across the scarp since its formation, rather than the elapsed time since a scarp-generating event like an earthquake.

That example was for just one relative age, 10 m^2. If we don't provide an `age` parameter it will search over a large range of ages from 0 to 3000 m^2.

[9]:
```
# Search over all ages in default range
# This can be slow on a laptop!
res = sl.match(data, Scarp, scale=100.)
```

Again, we'll do some masking to discard false positives and low-SNR features

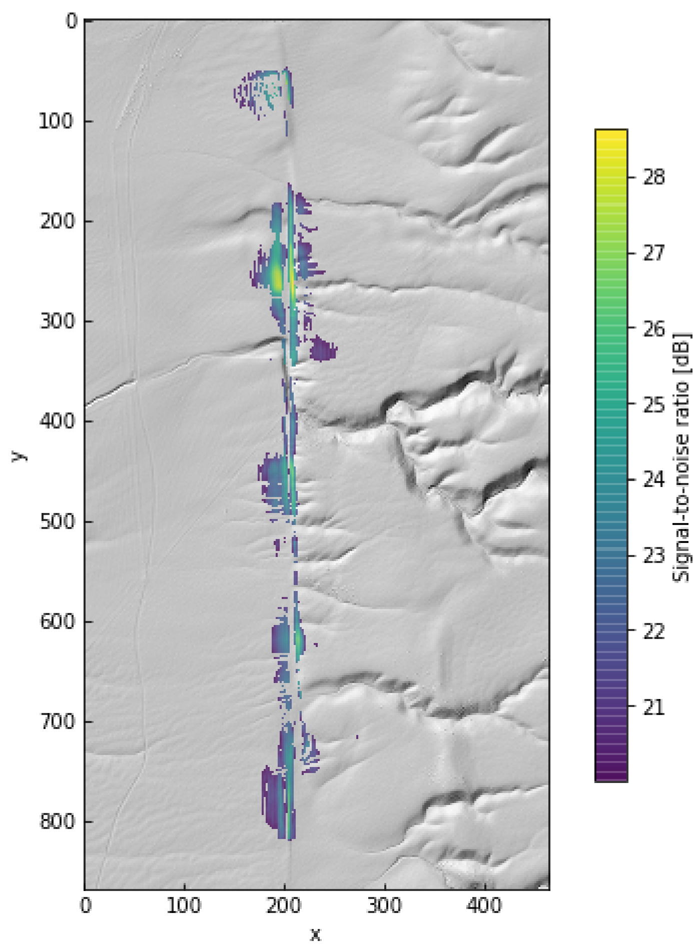
[10]:
```
angle, snr = [res[2], res[3]]
mask = snr < 100

# Mask out low SNR pixels
```

(continues on next page)

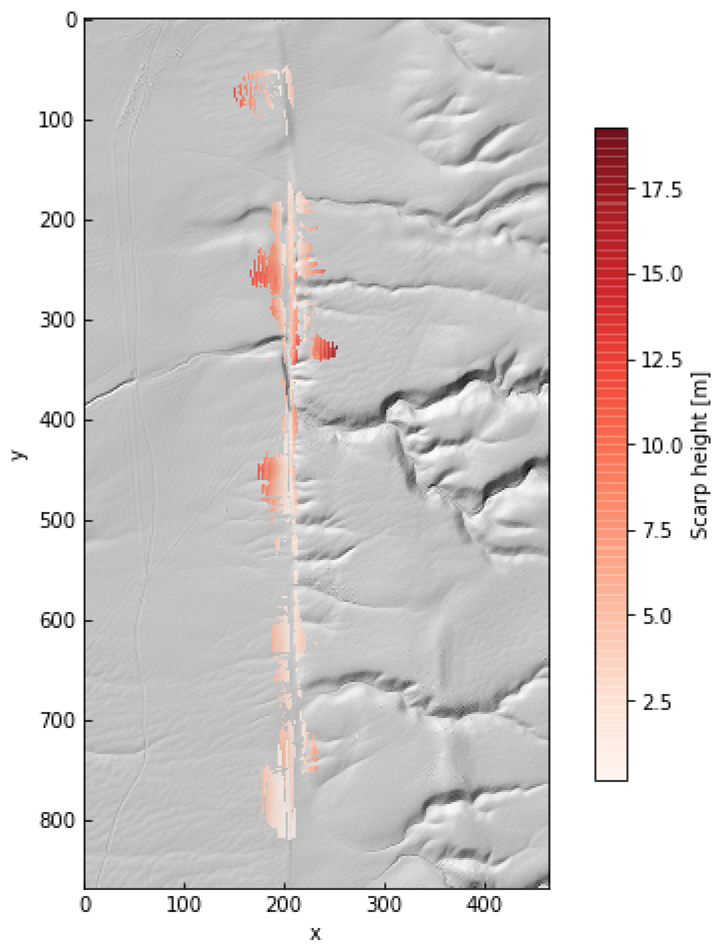
(continued from previous page)

```
res = np.array(res)
res[:, mask] = np.nan

# Mask out pixels with orientations far from vertical
ew = np.abs(angle) >= 5 * np.pi / 180.
res[:, ew] = np.nan

# Mask out pixels on edges of dataset (for roads)
res[:, :, 0:150] = np.nan
res[:, :, -150:] = np.nan

amp, age, angle, snr = res
amp = np.abs(amp)
```

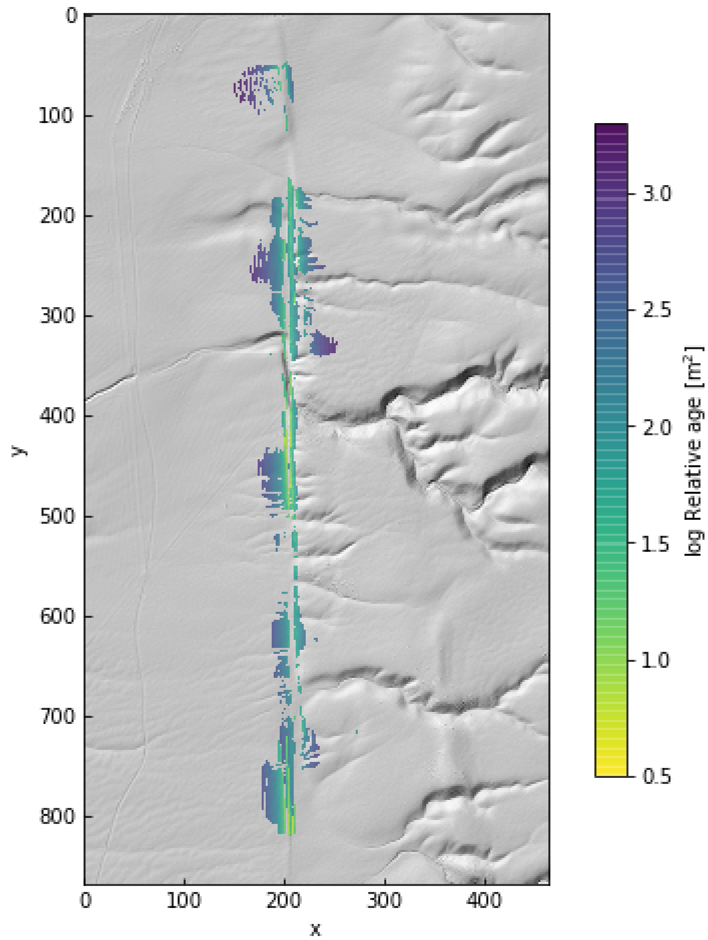
[11]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(10 * np.log10(res[3]), alpha=0.75, cmap='viridis')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Signal-to-noise ratio [dB]')
```



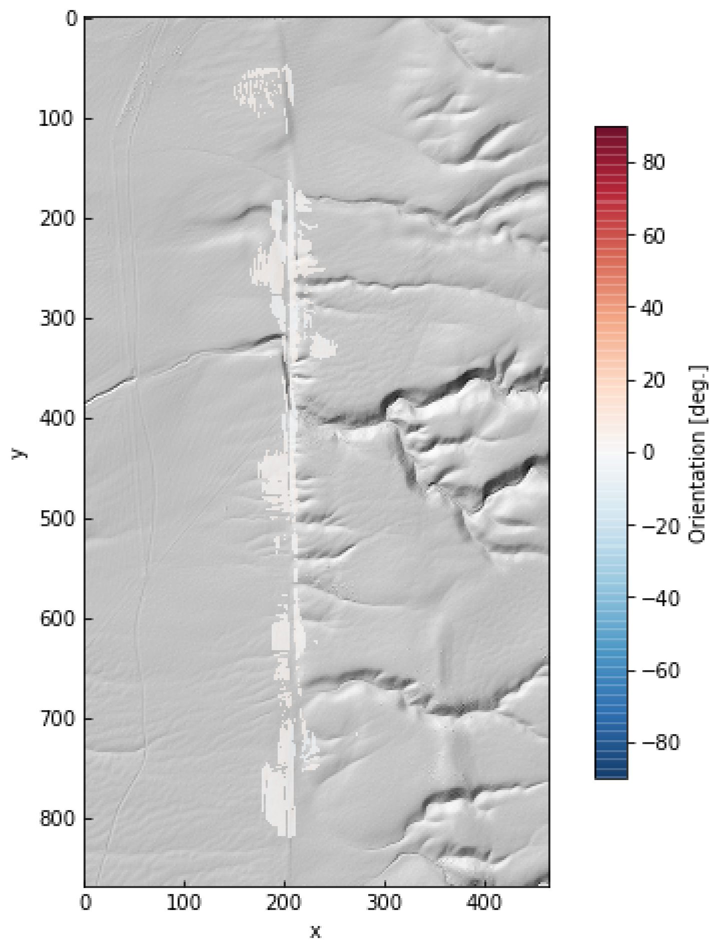
[12]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(amp, alpha=0.75, cmap='Reds')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Scarp height [m]')
```



[13]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(np.log10(age), alpha=0.75, cmap='viridis_r')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='log Relative age [m$^2$]')
```



[14]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(angle * 180 / np.pi, alpha=0.75, cmap='RdBu_r', vmin=-90, vmax=90)
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Orientation [deg.]')
```



The parameter grids are just Numpy arrays. This gives us the option of looking at along-strike variations in age or height.

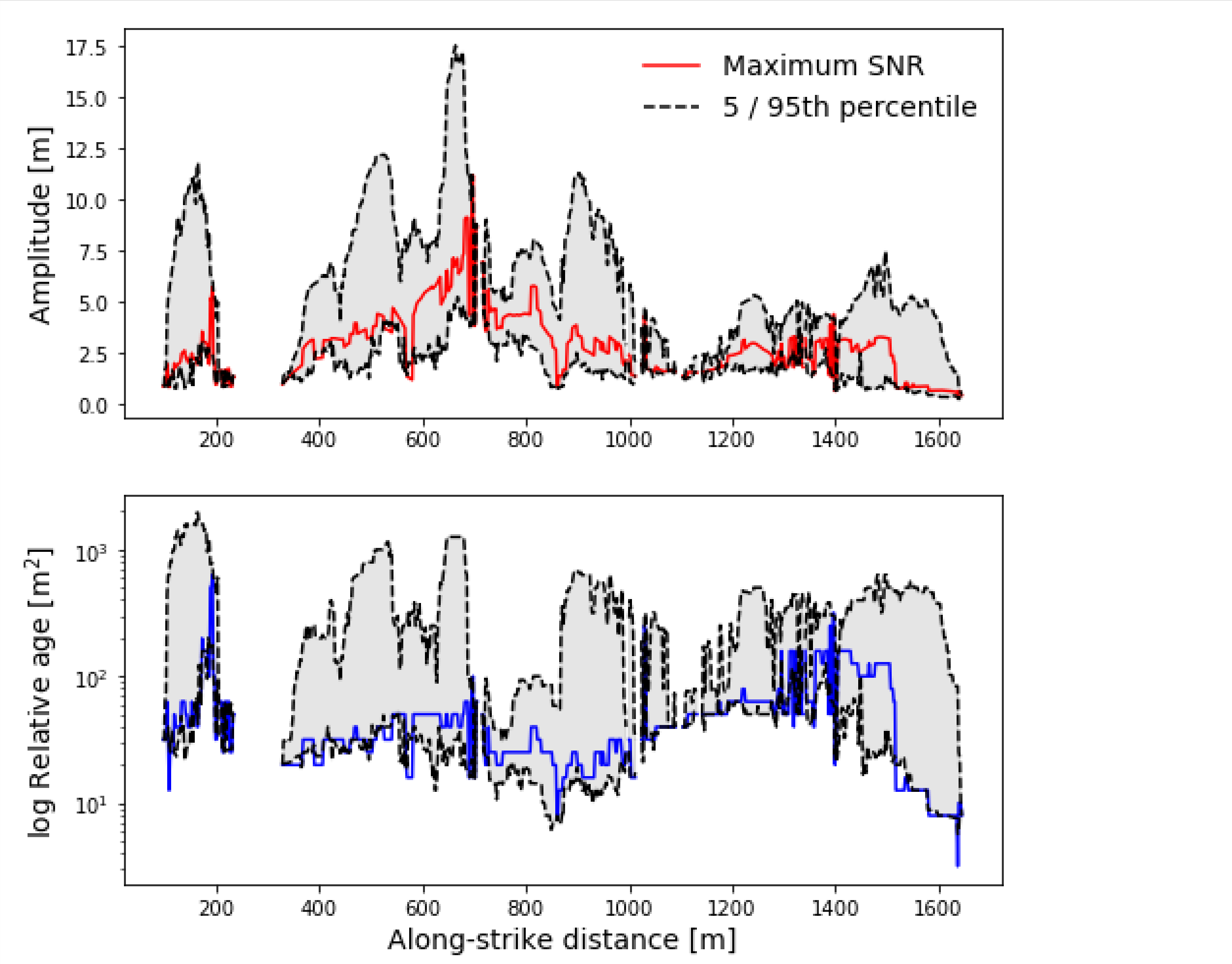
To do this, we iterate through the results to collect the maximum SNR pixels. A for loop isn't the most efficient way to do this, this is just for clarity!

[15]:
```
best_amps = []
best_ages = []
for i, row in enumerate(snr):
    idx = np.where(row == np.nanmax(row))[0]
    if len(idx) > 0:
        j = idx[0]
        best_amps.append(amp[i][j])
        best_ages.append(age[i][j])
    else:
        best_amps.append(np.nan)
        best_ages.append(np.nan)

best_amps = np.array(best_amps)
best_ages = np.array(best_ages)
```

Add some percentile bounds for each row to give a little context to the single-pixel estimates from maximum SNR.

[16]:
```
amp5 = [np.nanpercentile(row, 5) for row in amp]
amp95 = [np.nanpercentile(row, 95) for row in amp]

age5 = [np.nanpercentile(row, 5) for row in age]
age95 = [np.nanpercentile(row, 95) for row in age]
```

[17]:
```
fig, ax = plt.subplots(2, 1, figsize=(8, 8))
x = np.arange(amp.shape[0]) * dx

ax[0].fill_between(x, y1=amp5, y2=amp95, color='k', alpha=0.1)
ax[0].plot(x, best_amps, 'r-', label='Maximum SNR')
ax[0].plot(x, amp5, 'k--', label='5 / 95th percentile')
ax[0].plot(x, amp95, 'k--')
ax[0].set_ylabel('Amplitude [m]', fontsize=14)

ax[1].fill_between(x, y1=age5, y2=age95, color='k', alpha=0.1)
ax[1].plot(x, best_ages, 'b-')
ax[1].plot(x, age5, 'k--')
ax[1].plot(x, age95, 'k--')
ax[1].set(yscale='log')
ax[1].set_xlabel('Along-strike distance [m]', fontsize=14)
ax[1].set_ylabel('log Relative age [m$^2$]', fontsize=14)

leg = ax[0].legend(loc='upper right', frameon=False, fontsize=14)
```



We can see there's some variability and a gap around the position of Wallace Creek, at about 700 m. The gap occurs because we filtered pixels by orientation; in the channel itself, there are no pixels oriented at about 0 deg., which is the fault zone orientation in this sample dataset.

Since the fault is right-lateral in this case, one working model is that rightmost scarps were intially formed in earlier events, and the scarps closer to the creek formed more recently. As the scarps continue away from the bank of that channel, from 800 to 1400 m, they get noticeably smoother. This is captured in the gradient in the estimated ages at those locations.

Of course, this is predicated on each scarp resulting from a single surface offset. That's not usually the case, as multiple surface-rupturing earthquakes may revisit an area. This can lead to smaller subisidary slope breaks (multiple-event composite scarps) and generally complicate the interpretation of morphologic dating.

1.4 Extracting channels

This uses the Channel template to find channel network pixels by highlighting high-curvature parts of the landscape.

It is available as a Jupyter notebook (link) in the repository. Sample data is provided in the data folder.

1.4.1 Channel extraction in geomorphology

Wavelet analysis has been used to identify channels and rough landscape elements since the early days of high-resolution topographic data (e.g., Lashermes, et al., 2007). More recently, other approaches have become popular for extracting channel heads specifically. These include GeoNet, which uses nonlinear filtering and a multi-scale analysis of DEM curvature (Passalacqua, et al., 2010) and DrEICH, which identifies channel heads based on a fluvial-hillslope process transition encoded in elevation-flow length profiles (Clubb, et al., 2014).

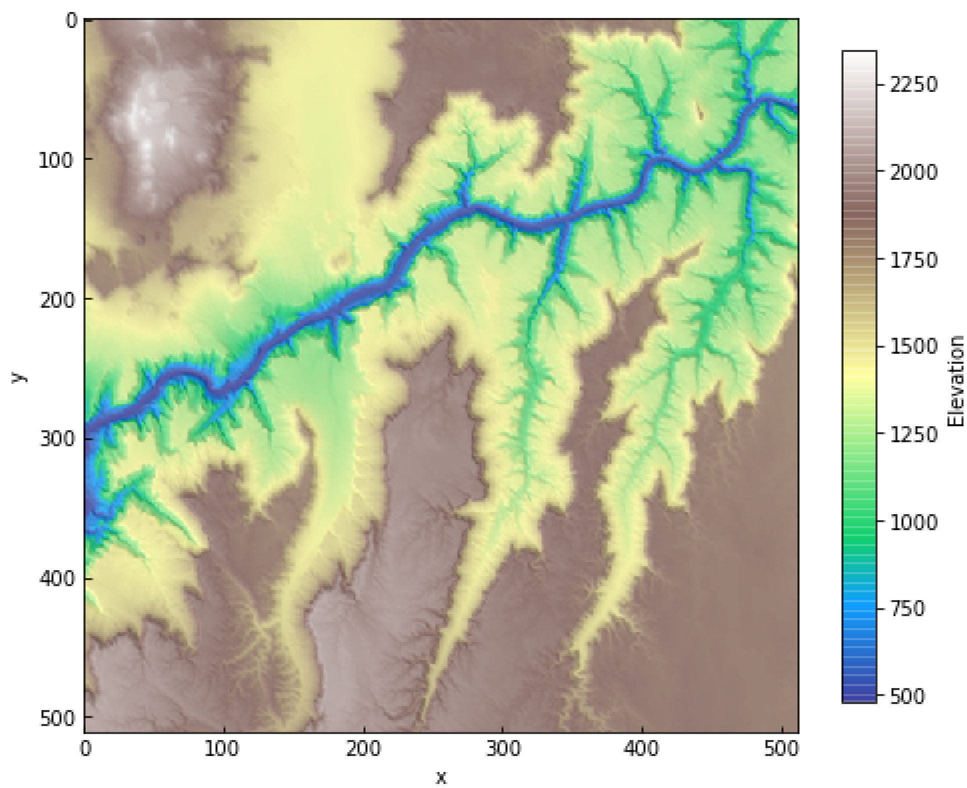
This example uses a Ricker wavelet similar to those used in earlier work to estimate channel or valley depth and orientation. Unlike the radially symmetric wavelets Lashermes, et al., 2007 or other approaches, this is a windowed version of that function that is linear in one direction.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
import scarplet as sl
from scarplet.datasets import load_grandcanyon
from scarplet.WindowedTemplate import Scarp, Channel
```

This sample data is an SRTM tile including part the Grand Canyon.

[3]:
```
data = load_grandcanyon()
data.plot(color=True, figsize=(8,8))
```



SRTM data is coarse – and in this case, we are working with ~76 m resolution (this is a tile at Web Mercator zoom level 10). The range of resolvable curvature will be very low. We can change this to work pixel units instead.

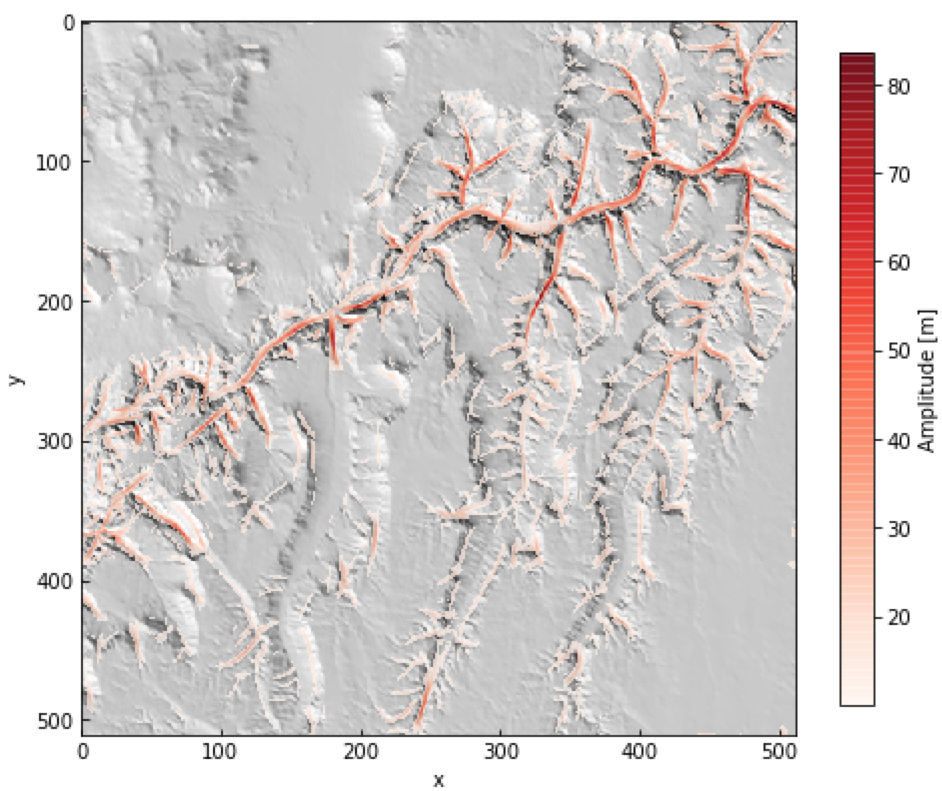
[4]:
```
data._georef_info.dx = 1.
data._georef_info.dy = -1.
```

[5]:
```
params = {'scale': 10.,
          'age': 0.1,
          'ang_min': -np.pi / 2,
          'ang_max': np.pi / 2
         }

res = sl.match(data, Channel, **params)
```

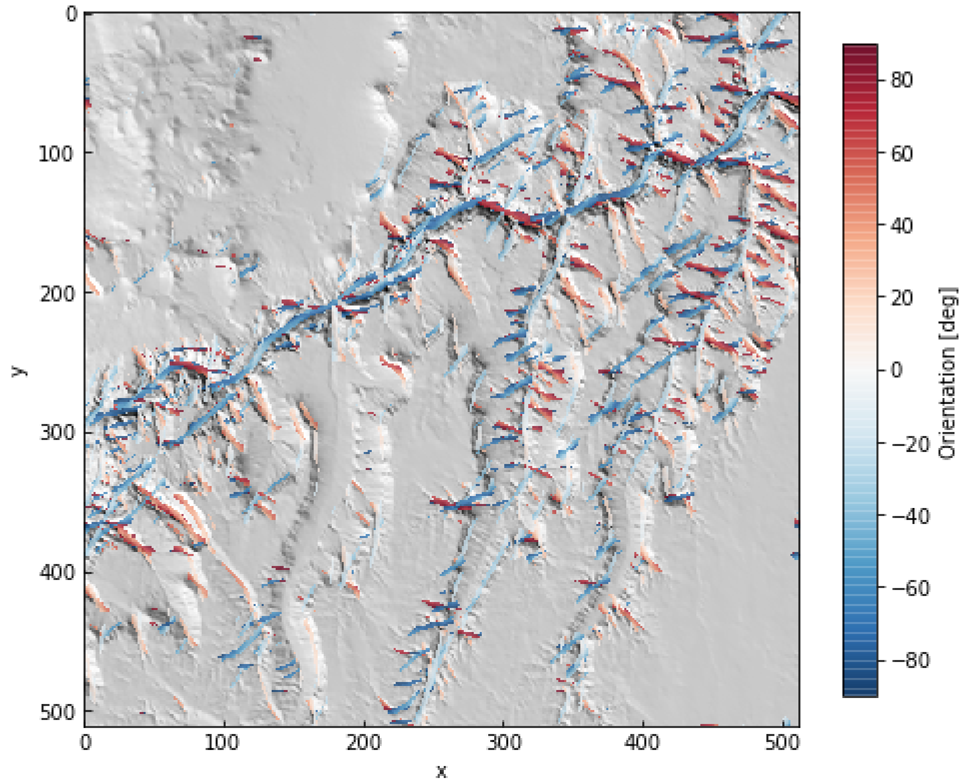
In this case, using the Ricker wavelet, negative amplitudes correspond to ridges and other convexities. Let's discard pixels with low amplitudes to see the main channels in the network.

[7]:
```
mask = res[0] < 10.
res[:, mask] = np.nan
```

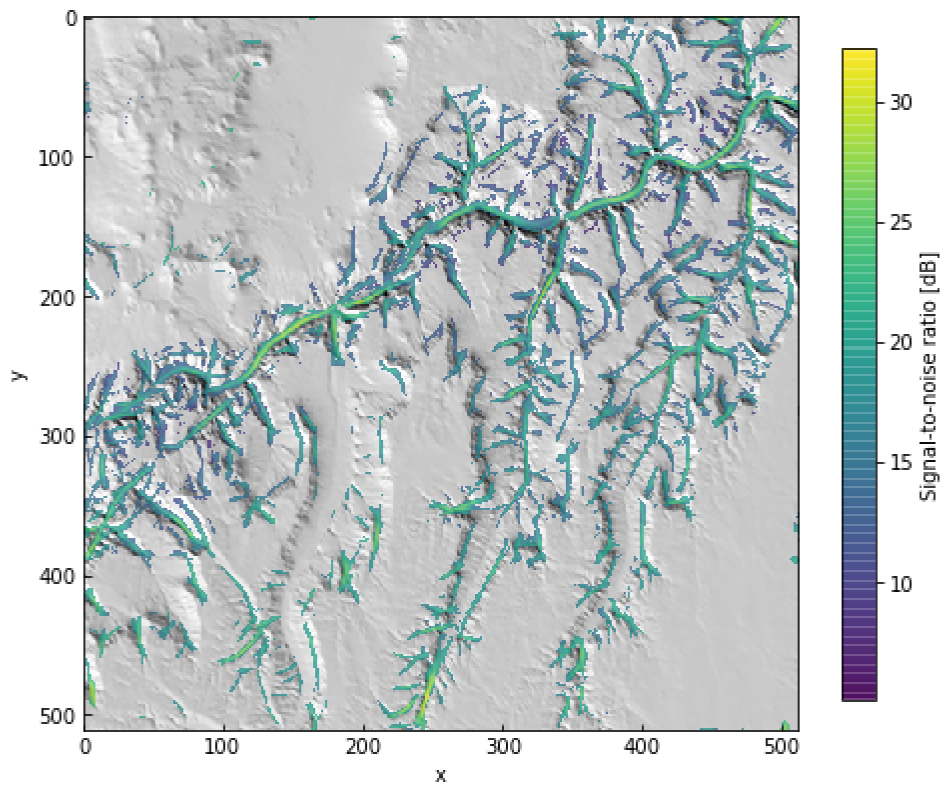
[13]:
```
data = sl.datasets.load_grandcanyon()
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(res[0], alpha=0.75, cmap='Reds')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Amplitude [m]')
```



[14]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
angle = res[2] * 180. / np. pi
im = ax.imshow(angle, alpha=0.75, cmap='RdBu_r')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Orientation [deg]')
```



[15]:
```
data.plot(color=False, figsize=(8, 8))
ax = plt.gca()
im = ax.imshow(10 * np.log10(res[3]), alpha=0.75, cmap='viridis')
cb = plt.colorbar(im, ax=ax, shrink=0.75, label='Signal-to-noise ratio [dB]')
```



1.5 Multiprocessing and scarplet

This simple example shows how to use the `match_template` and `compare` methods with a multiprocessing worker pool.

It is available as a Jupyter notebook (link) in the repository. Sample data is provided in the data folder.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from functools import partial
from multiprocessing import Pool

import scarplet as sl
from scarplet.datasets import load_synthetic
from scarplet.WindowedTemplate import Scarp
```

[2]:
```
data = load_synthetic()
```

[3]:
```
# Define parmaters for search
scale = 10
age = 10.
angles = np.linspace(-np.pi / 2, np.pi / 2, 181)
nprocs = 3
```

For each set of input parameters, we can start a separate masking task. These can be run in parallel, which is what `scarplet` does by default.

[4]:
```
# Start separate search tasks
pool = Pool(processes=nprocs)
wrapper = partial(sl.match_template, data, Scarp, scale, age)
results = pool.imap(wrapper, angles, chunksize=1)
```

[5]:
```
%%time
# Reduce the final results as they are completed
ny, nx = data.shape
best = sl.compare(results, nx, ny)
```

```
CPU times: user 720 ms, sys: 296 ms, total: 1.02 s
Wall time: 2.48 s
```

To compare, we can a loop to fit the templates sequentially.

[6]:
```
%%time
best = np.zeros((4, ny, nx))
for angle in angles:
    results = sl.match_template(data, Scarp, scale, age, angle)
    best = sl.compare([best, results], nx, ny)
```

```
CPU times: user 3.76 s, sys: 708 ms, total: 4.47 s
Wall time: 3.62 s
```

We get a fairly good speed up just using three processes on this small test case. Distributing tasks and reducing results using a cluster can make processing large datasets feasible. For example, dask provides nice distributed task management in Python.

1.6 API Reference

This package is structured so that most functions are implemented in a `core` submodule and templates are defined as subclasses of `WindowedTemplate` in the `WindowedTemplate` submodule. Spatial data and I/O is handled by classes defined in `dem`.

1.6.1 Core functionality

scarplet.core module

Functions for determining best-fit template parameters by convolution with a grid

`scarplet.core.`**`calculate_amplitude`**(*dem*, *Template*, *scale*, *age*, *angle*)

Calculate amplitude and SNR of features using a template

Parameters

dem [DEMGrid] Grid object of elevation data

Template [WindowedTemplate] Class representing template function

scale [float] Scale of template function in DEM cell units

age [float] Age parameter for template function

angle [float] Orientation of template in radians

Returns

amp [np.array] 2-D array of amplitudes for each DEM pixel

snr [np.array] 2-D array of signal-to-noise ratios for each DEM pixel

`scarplet.core.`**`calculate_best_fit_parameters`**(*dem, Template, scale, age, ang_max=<MagicMock name='mock.__truediv__()' id='140376671519576'>, ang_min=<MagicMock name='mock.__neg__().__truediv__()' id='140376671612880'>, **kwargs*)

Calculate best-fitting parameters using a template with parallel search

Parameters

dem [DEMGrid] Grid object of elevation data

Template [WindowedTemplate] Class representing template function

scale [float] Scale of template function in DEM cell units

age [float] Age parameter for template function

Returns

results [np.array] Array of best amplitudes, ages, orientations, and signal-to-noise ratios for each DEM pixel. Dimensions of (4, height, width).

Other Parameters

ang_max [float, optional] Maximum orietnation of template, default pi / 2

ang_min [float, optional] Minimum orietnation of template, default -pi / 2

`scarplet.core.`**`calculate_best_fit_parameters_serial`**(*dem, Template, scale, ang_max=<MagicMock name='mock.__truediv__()' id='140376660332328'>, ang_min=<MagicMock name='mock.__neg__().__truediv__()' id='140376661312512'>, **kwargs*)

Calculate best-fitting parameters using a template

Parameters

dem [DEMGrid] Grid object of elevation data

Template [WindowedTemplate] Class representing template function

scale [float] Scale of template function in DEM cell units

Returns

best_amp [np.array] 2-D array of best-fitting amplitudes for each DEM pixel

best_age [np.array] 2-D array of best-fitting agees for each DEM pixel

best_angle [np.array] 2-D array of best-fitting orientations for each DEM pixel

best_snr [np.array] 2-D array of maximum signal-to-noise ratios for each DEM pixel

Other Parameters

ang_max [float, optional] Maximum orietnation of template, default pi / 2

ang_min [float, optional] Minimum orietnation of template, default -pi / 2

kwargs [optional] Any additional keyword arguments that may be passed to the template() method of the Template class

`scarplet.core.`**`compare`**(*results*, *ny*, *nx*)

Compare template matching results from asynchronous tasks

Parameters

results [iterable] Iterable containing outputs of a template matching method

ny [int] Number of rows in output

nx [int] Number of columns in output

Returns

best_amp [np.array] 2-D array of best-fitting amplitudes

best_age [np.array] 2-D array of best-fitting morphologic ages

best_angle [np.array] 2-D array of best-fitting orientations

best_snr [np.array] 2-D array of maximum signal-to-noise ratios

`scarplet.core.`**`load`**(*filename*)

Load DEM from file

Parameters

filename [string] Filename of DEM

Returns

data_obj [DEMGrid] DEMGrid object with DEM data

`scarplet.core.`**`match`**(*data*, *Template*, ***kwargs*)

Match template to input data from DEM

Parameters

data [DEMGrid] DEMGrid object containing input data

Template [WindowedTemplate] Class of template function to use

Returns

results [np.array] Array of best amplitudes, ages, orientations, and signal-to-noise ratios for each DEM pixel. Dimensions of (4, height, width).

`scarplet.core.`**`match_template`**(*data*, *Template*, *scale*, *age*, *angle*, ***kwargs*)

Match template function to curvature using convolution

Parameters

data [DEMGrid] Grid object of elevation data

Template [WindowedTemplate] Class representing template function

scale [float] Scale of template function in DEM cell units

age [float] Age parameter for template function

angle [float] Orientation of template in radians

Returns

amp [np.array] 2-D array of amplitudes for each DEM pixel

age [np.array] template age in m2

angle [np.array] template orientation in radians

snr [np.array] 2-D array of signal-to-noise ratios for each DEM pixel

Other Parameters

kwargs [optional] Any additional keyword arguments that may be passed to the template() method of the Template class

References

Modifies method described in

Hilley, G.E., DeLong, S., Prentice, C., Blisniuk, K. and Arrowsmith, J.R., 2010. Morphologic dating of fault scarps using airborne laser swath mapping (ALSM) data. Geophysical Research Letters, 37(4). https://dx.doi.org/10.1029/2009GL042044

`scarplet.core.plot_results` (*data*, *results*, *az=315*, *elev=45*, *figsize=(4, 16)*)

Plots maps of results from template matching

Parameters

data [DEMGrid] DEMGrid object containing input data

results [np.array] Array of best-fitting results from compare() or similar function

1.6.2 Templates

scarplet.WindowedTemplate module

Class for windowed template matching over a spatial grid

class `scarplet.WindowedTemplate.Channel` (*d*, *f*, *alpha*, *nx*, *ny*, *de*)

Bases: *scarplet.WindowedTemplate.Ricker*

Duplicate class for Ricker wavelet used for fluvial channels

Methods

`template()`	Template function for windowed Ricker wavelet

get_coordinates	
get_mask	
get_window_limits	

class `scarplet.WindowedTemplate.Crater` (*r*, *kt*, *nx*, *ny*, *de*)

Bases: *scarplet.WindowedTemplate.WindowedTemplate*

Template for radially symmetric crater

Attributes

r [float] Radius of crater in pixels

kt [float] Morphologic age of template crater rim in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

`template`()	Template function for radially symmetric crater

get_coordinates	
get_mask	
get_window_limits	

template()

Template function for radially symmetric crater

Returns

W [numpy array] Windowed template function

class scarplet.WindowedTemplate.**LeftFacingUpperBreakScarp**(*d*, *kt*, *alpha*, *nx*, *ny*, *de*)

Bases: `scarplet.WindowedTemplate.Scarp`

Template for upper slope break of vertical scarp (left-facing)

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

kt [float] Morphologic age of template in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

get_error_mask():	Return mask array that masks the lower slope break of scarp

get_err_mask()

Return mask array masking the lower half of scarp

Returns

mask [numpy array] Mask array for lower hald of scarp

class scarplet.WindowedTemplate.**Ricker**(*d*, *f*, *alpha*, *nx*, *ny*, *de*)

Bases: `scarplet.WindowedTemplate.WindowedTemplate`

Template using 2D Ricker wavelet

References

This implements a Ricker wavelet similar to thet used in the following work

Lashermes, B., Foufoula-Georgiou, E., and Dietrich, W. E., 2007, Channel network extraction from high resolution topography using wavelets. Geophysical Research Letters, 34(23). https://doi.org/10.1029/2007GL031140

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

kt [float] Morphologic age of template in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

template():	Returns array of windowed template function

get_window_limits()

template()

Template function for windowed Ricker wavelet

Returns

W [numpy array] Windowed template function

class scarplet.WindowedTemplate.**RightFacingUpperBreakScarp**(*d*, *kt*, *alpha*, *nx*, *ny*, *de*)

Bases: *scarplet.WindowedTemplate.Scarp*

Template for upper slope break of vertical scarp (right-facing)

Overrides template function to correct facign direction

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

kt [float] Morphologic age of template in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

get_error_mask():	Return mask array that masks the lower slope break of scarp
template():	Returns array of windowed template function

get_err_mask()

Return mask array masking the lower half of scarp

Returns

mask [numpy array] Mask array for lower half of scarp

template()

Return template function (uses numexpr where possible)

Returns

W [numpy array] Windowed template function

class scarplet.WindowedTemplate.**Scarp**(*d*, *kt*, *alpha*, *nx*, *ny*, *de*)

Bases: *scarplet.WindowedTemplate.WindowedTemplate*

Curvature template for vertical scarp

References

Adapted from template derived in

Hilley, G.E., DeLong, S., Prentice, C., Blisniuk, K. and Arrowsmith, J.R., 2010. Morphologic dating of fault scarps using airborne laser swath mapping (ALSM) data. Geophysical Research Letters, 37(4). https://dx.doi.org/10.1029/2009GL042044

Based on solutions to the diffusion equation published in

Hanks, T.C., 2000. The age of scarplike landforms from diffusion-equation analysis. Quaternary geochronology, 4, pp.313-338.

and many references therein.

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

kt [float] Morphologic age of template in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

template():	Returns array of windowed template function
template_numexpr():	Returns array of windowed template function optimized using numexpr

template()

Return template function

Returns

W [numpy array] Windowed template function

template_numexpr()

Return template function (uses numexpr where possible)

Returns

W [numpy array] Windowed template function

class scarplet.WindowedTemplate.**ShiftedLeftFacingUpperBreakScarp**(**args*, ***kwargs*)

Bases: *scarplet.WindowedTemplate.ShiftedTemplateMixin*, *scarplet.WindowedTemplate.LeftFacingUpperBreakScarp*

Methods

get_err_mask()	Return mask array masking the lower half of scarp
set_offset(dx, dy)	Set offset values
shift_template(W, dx, dy)	Shift template
template()	Template function for shifted template
template_numexpr()	Return template function (uses numexpr where possible)

get_coordinates	
get_mask	
get_window_limits	

class scarplet.WindowedTemplate.**ShiftedRightFacingUpperBreakScarp**(**args*, ***kwargs*)

Bases: *scarplet.WindowedTemplate.ShiftedTemplateMixin*, *scarplet.WindowedTemplate.RightFacingUpperBreakScarp*

Methods

get_err_mask()	Return mask array masking the lower half of scarp
set_offset(dx, dy)	Set offset values
shift_template(W, dx, dy)	Shift template
template()	Template function for shifted template
template_numexpr()	Return template function (uses numexpr where possible)

get_coordinates	
get_mask	
get_window_limits	

class scarplet.WindowedTemplate.**ShiftedTemplateMixin**(**args*, ***kwargs*)

Bases: *scarplet.WindowedTemplate.WindowedTemplate*

Mix-in for template that is offset from the window center

Overrides template function to shift template

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

kt [float] Morphologic age of template in m2

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

dx [float] X Offset of template center in data projection units

dy [float] Y Offset of template center data projection units

Methods

set_offset(dx, dy):	Set offset attrivutes t odx and dy
shift_template(W, dx, dy):	Shift template array W by dx and dy
template():	Returns array of windowed template function

set_offset(*dx*, *dy*)
Set offset values

Parameters

dx [float] X Offset of template center in data projection units

dy [float] Y Offset of template center data projection units

shift_template(*W*, *dx*, *dy*)
Shift template

Parameters

W [numpy array] Windowed template function

dx [float] X Offset of template center in data projection units

dy [float] Y Offset of template center data projection units

Returns

W [numpy array] Shifted windowed template function

template()
Template function for shifted template

Returns

W [numpy array] Shifted windowed template function

class scarplet.WindowedTemplate.**WindowedTemplate**
Bases: `object`

Base class for windowed template function

Attributes

d [float] Scale of windowed template function in data projection units

alpha [float] Orientation of windowed template function in radians

c [float] Curvature limit of template

nx [int] Number of columns in template array

ny [int] Number of rows in template array

de [float] Spacing of template grid cells in dat projection units

Methods

get_coordinates():	Get arrays of coordinates for template grid points
get_mask():	Get mask array giving curvature extent of template window
get_window_limits():	Get mask array giving window extent

get_coordinates()

get_mask()

get_window_limits()

1.6.3 Data and IO

scarplet.dem module

Classes for loading digital elevation models as numeric grids

class scarplet.dem.**BaseSpatialGrid**(*filename=None*)

Bases: *scarplet.dem.GDALMixin*

Base class for spatial grid

Methods

dtype	
is_contiguous(grid)	Returns true if grids are contiguous or overlap
load(filename)	Load grid from file
merge(grid)	Merge this grid with another BaseSpatialGrid.
plot(**kwargs)	Plot grid data
save(filename)	Save grid as georeferenced TIFF

dtype = <MagicMock name='mock.GDT_Float32' id='140376660782104'>

is_contiguous(*grid*)

Returns true if grids are contiguous or overlap

Parameters

grid [BaseSpatialGrid]

load(*filename*)

Load grid from file

merge(*grid*)

Merge this grid with another BaseSpatialGrid.

Wrapper argound gdal_merge.py.

Parameters

grid [BaseSpatialGrid]

Returns

merged_grid [BaseSpatialGrid]

plot (***kwargs*)
Plot grid data

Keyword args: Any valid keyword argument for matplotlib.pyplot.imshow

save (*filename*)
Save grid as georeferenced TIFF

class scarplet.dem.**CalculationMixin**
Bases: `object`

Mix-in class for grid calculations

class scarplet.dem.**DEMGrid** (*filename=None*)
Bases: *scarplet.dem.CalculationMixin*, *scarplet.dem.BaseSpatialGrid*

Class representing grid of elevation values

Methods

dtype	
is_contiguous(grid)	Returns true if grids are contiguous or overlap
load(filename)	Load grid from file
merge(grid)	Merge this grid with another BaseSpatialGrid.
plot([color])	Plot grid data
save(filename)	Save grid as georeferenced TIFF

plot (*color=True*, ***kwargs*)
Plot grid data

Keyword args: Any valid keyword argument for matplotlib.pyplot.imshow

class scarplet.dem.**GDALMixin**
Bases: `object`

class scarplet.dem.**GeorefInfo**
Bases: `object`

class scarplet.dem.**Hillshade** (*dem*)
Bases: *scarplet.dem.BaseSpatialGrid*

Class representing hillshade of DEM

Methods

dtype	
is_contiguous(grid)	Returns true if grids are contiguous or overlap
load(filename)	Load grid from file
merge(grid)	Merge this grid with another BaseSpatialGrid.
plot([az, elev])	Plot hillshade
save(filename)	Save grid as georeferenced TIFF

plot (*az=315*, *elev=45*)
Plot hillshade

scarplet.datasets package

Submodules

scarplet.datasets.base module

Convenience functions to load example datasets

scarplet.datasets.base.**load_carrizo**()

Load sample dataset containing fault scarps along the San Andreas Fault from the Wallace Creek section on the Carrizo Plain, California, USA

Data downloaded from OpenTopography and collected by the B4 Lidar Project: https://catalog.data.gov/dataset/b4-project-southern-san-andreas-and-san-jacinto-faults

scarplet.datasets.base.**load_grandcanyon**()

Load sample dataset containing part of channel network in the Grand Canyon Arizona, USA

Data downloaded from the Terrain Tile dataset, part of Amazon Earth on AWS https://registry.opendata.aws/terrain-tiles/

scarplet.datasets.base.**load_synthetic**()

Load sample dataset of synthetic fault scarp of morphologic age 10 m2

Python Module Index

S

Index

T

W