pyosp

Release 0.1.7

PyOSP developers

Feb 05, 2022

GETTING STARTED

PyOSP (Python Object-oriented Swath Profile) is an intelligent swath tool that can characterize complex topographic features. Unlike traditional swath methods have been limited to rectangular or simplified curvilinear sampling blocks, PyOSP can objectively characterize complex geomorphic boundaries using elevation, slope angle, topographic position index (TPI), or other raster calculations by intelligently assimilating geo-processing information into swath analysis.

PyOSP supports Python 3.6 or higher, and depends on Numpy , Matplotlib , GDAL , SciPy, and Shapely .

CHAPTER

ONE

INSTALLATION

1.1 Installing with Anaconda / conda

The easiest way to install PyOSP is through conda with the following command:

```
conda install -c conda-forge pyosp
```

1.2 Installing in a new environment (recommended)

Although it is not required, installing the library in a clean environment represents good practice and helps avoid potential dependency conflicts. We also recommends install all dependencies with pyosp through conda-forge channel:

```
conda create -n env_pyosp
conda activate env_pyosp
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict
conda install python=3 pyosp
```

1.3 Installing from source

You may install the development version by cloning the source repository and installed locally:

```
git clone https://github.com/PyOSP-devs/PyOSP.git
cd pyosp
conda install --file requirements.txt
python setup.py install
```

CHAPTER

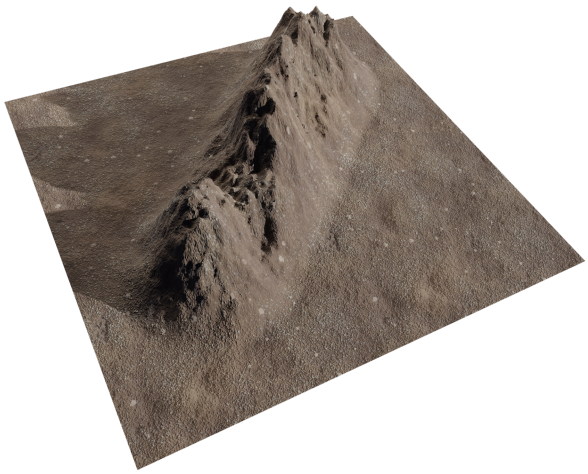
TWO

FIXED-WIDTH CURVILINEAR SWATH PROFILE

2.1 At the beginning

Fixed-width curvilinear swath profile is equivalent to traditional swath profile that typically stacks a set of profile lines perpendicular to a baseline within a swath shape, or band, from which the elevation or other types of data (e.g., precipitation, relief, roughness, etc.) are sampled. Runing fixed-width swath analysis in PyOSP follows simple workflow. Here, we use a synthetic case to illustrate the implementation steps.

Figure below shows an artificial mountain range with a homogeneous width (90m) situated above the flat ground (0m).



As suggested by its name, the fixed-width swath analysis requires a defined width which is set as 100 m here in order to encompass the cross width of the mountain (90 m). Additionally, the baseline is set across the longitudinal axis of the mountain. The following shows the code snippet to generate a swath object.

2.2 Generate a swath object

[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

orig = pyosp.Orig_curv(baseline, raster, width=100,
```

(continues on next page)

(continued from previous page)

```
                        line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

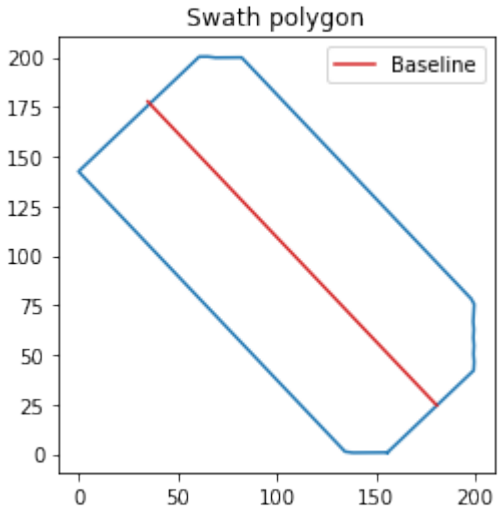
2.3 Swath polygon and polylines

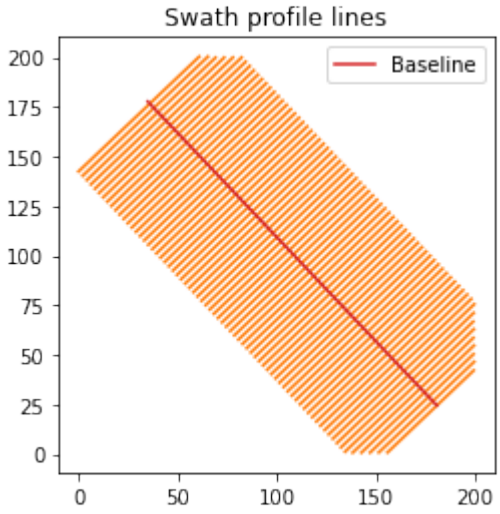
Using methods *out_polygon()* or *out_polylines()*, we can generate shapely objects to be plotted with Matplotlib

PyOSP also provides function *read_shape* to let us read the baseline shapefile into shapely object, and plot with polygon and polylines as below:

[2]:
```
import matplotlib.pyplot as plt
from pyosp import read_shape

# read the baseline shape
line_shape = read_shape(baseline)
lx, ly = line_shape.xy

# Plot the swath polygon
fig, ax = plt.subplots()
swath_polygon = orig.out_polygon()
px, py = swath_polygon.exterior.xy
ax.plot(px, py)
ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath polygon")
ax.legend()

# Plot the swath profile lines
fig, ax = plt.subplots()
swath_polylines = orig.out_polylines()
for line in swath_polylines:
    x, y = line.xy
    ax.plot(x, y, color='C1')

ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath profile lines")
ax.legend()
```

[2]:
```
<matplotlib.legend.Legend at 0x7fa4aa7d8700>
```





Noticing that the polygon is generated by connecting ending points of polylines. In cases that polylines are heavily overlapping with each other, **the presented swath polygon could be less reliable**. It is worth noting that data is always sampled through the locations of polylines, rather than polygon.

The above also shows that there are cutting conors around the swath area. Now, let's investigate this by plotting together with raster dataset.

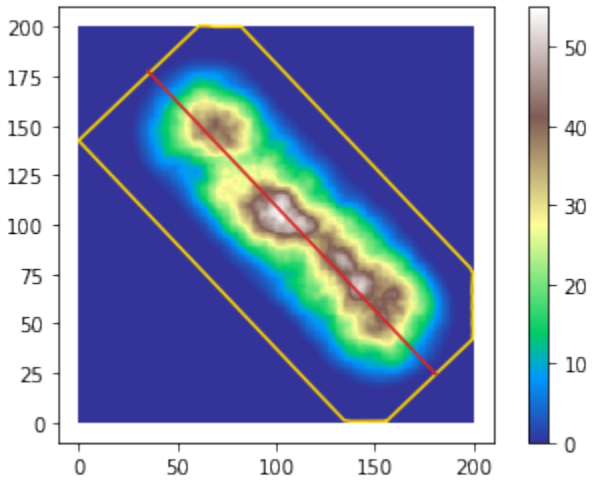
[3]:
```
import numpy as np
import gdal

ds = gdal.Open(raster)
data = ds.ReadAsArray()
gt = ds.GetGeoTransform()

xRes = gt[1]
yRes = gt[5]

# get the edge coordinates and add half the resolution
# to go to center coordinates
```

(continues on next page)

(continued from previous page)

```
xmin = gt[0] + xRes * 0.5
xmax = gt[0] + (xRes * ds.RasterXSize) - xRes * 0.5
ymin = gt[3] + (yRes * ds.RasterYSize) + yRes * 0.5
ymax = gt[3] - yRes * 0.5

ds = None

# create a grid of xy coordinates
xy = np.mgrid[xmin:xmax+xRes:xRes, ymax+yRes:ymin:yRes]

fig, ax = plt.subplots()
ax.plot(px, py, color="gold", label='swath polygon')
im = ax.pcolormesh(xy[0], xy[1], data.T, cmap=plt.cm.terrain)
ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_xlim([-10, 210])
ax.set_ylim([-10, 210])
plt.colorbar(im)
```

[3]:
```
<matplotlib.colorbar.Colorbar at 0x7fa4aa5463a0>
```



As shown above, the swath area was bounded by the range of raster dataset. During the calculation of swath object, PyOSP keeps detecting the boundary of dataset to prevent sampling into no data region.

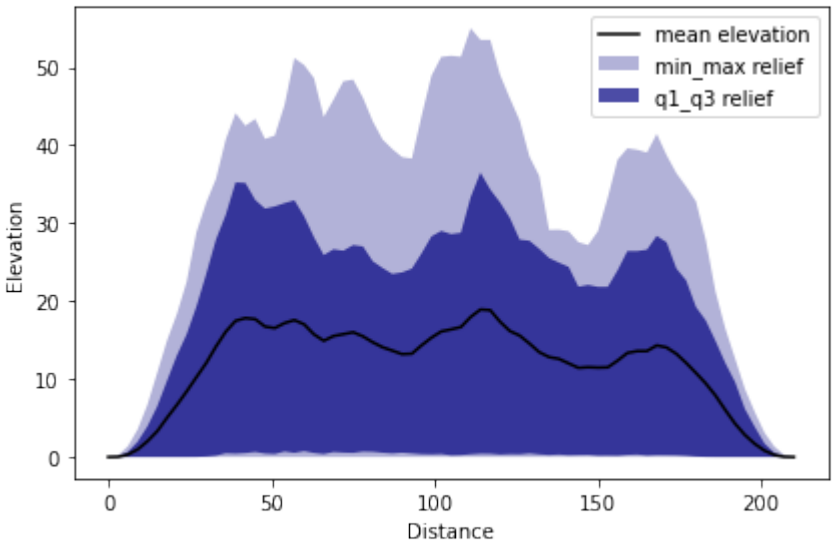
PyOSP can also export generated polygon or polylines as shapefiles. Hence the result can be read and processed in other GIS softwares like ArcGIS, QGIS, etc. For example:

[4]:
```
# This will generate polygon and polyline shapefiles in the current folder
pyosp.write_polygon(orig.out_polygon(), "./orig_polygon.shp")
pyosp.write_polylines(orig.out_polylines(), "./orig_polylines.shp")
```

2.4 Swath profile

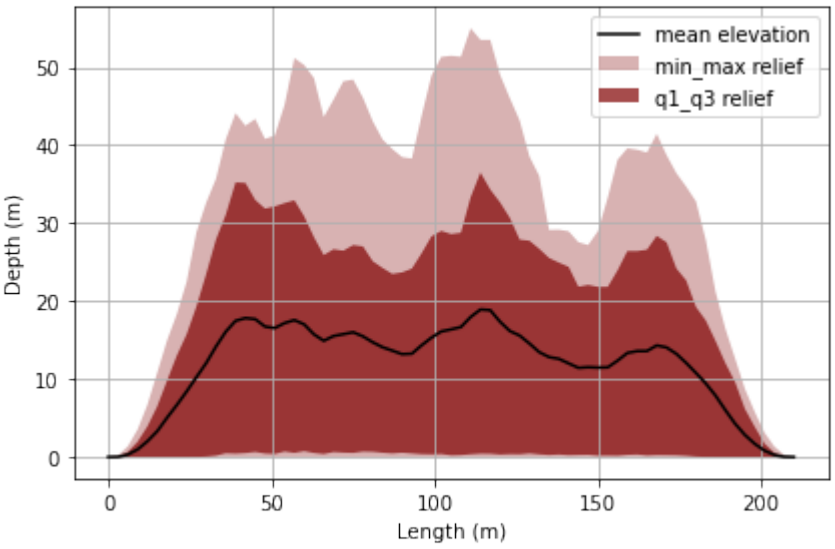
To plot the corresponding swath profile, simply run method:

[5]:
```
orig.profile_plot()
```



If the figure needs to be customized, an *axis* argument can be passed to the object:

[6]:
```
fig, ax = plt.subplots()
orig.profile_plot(ax=ax, color="maroon")
ax.set_xlabel("Length (m)")
ax.set_ylabel("Depth (m)")
ax.grid()
```



2.5 Summary

This section covers the basic usage of PyOSP to perform traditional fixed width swath profile. For more examples and advance topics, please refer to our *example gallery*.

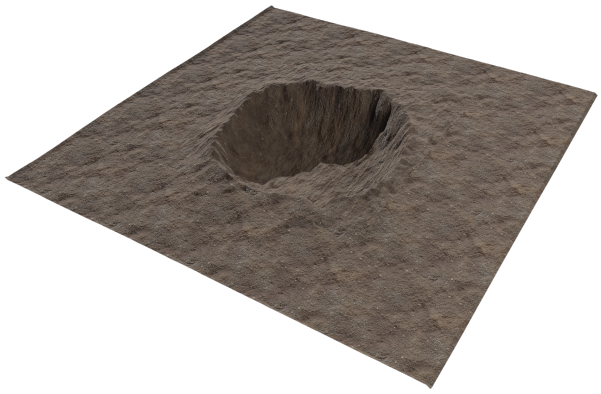
CHAPTER

THREE

FIXED-RADIUS CIRCULAR SWATH PROFILE

3.1 At the beginning

Similar to fixed-width curvilinear swath profile, fixed-radius swath profile here refers to traditional circular swath profile. It usually starts from defining a central point, then profile-lines are stacked along radial directions outwards until a certain length is reached. To create a fixed-radius circular swath object, `Orig_cir` is the class to call.



3.2 Generate a swath object

[2]:
```
import pyosp

center = pyosp.datasets.get_path("center.shp")  # path to the central point shapefile
raster = pyosp.datasets.get_path("crater.tif")  # path to the raster tif

orig = pyosp.Orig_cir(center, raster, radius=80,
                      ng_start=0, ng_end=360,
                      ng_stepsize=5, radial_stepsize=None)
```

```
Processing: [########################] 72 of 72 lineSteps
```

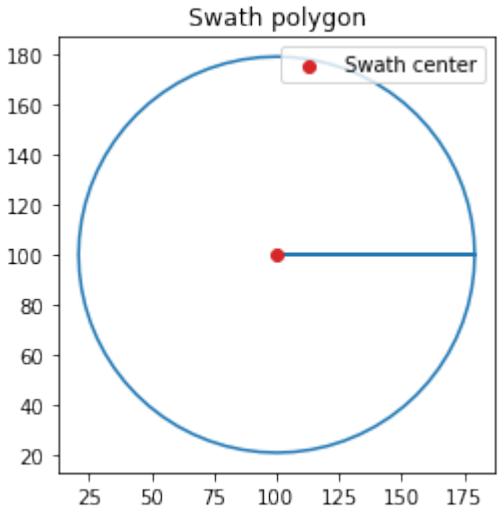
3.3 Swath polygon and polylines

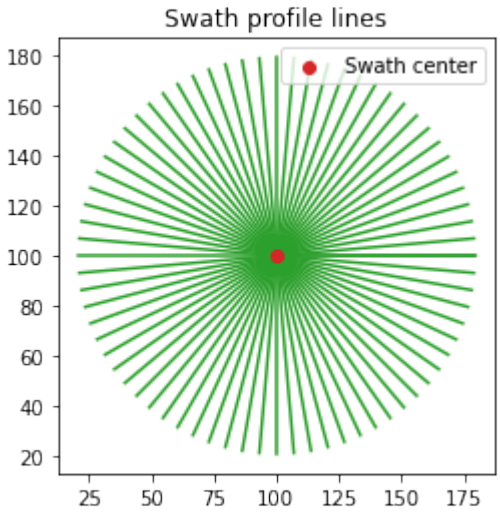
Using methods *out_polygon()* or *out_polylines()*, we can generate shapely objects to be plotted with Matplotlib.

PyOSP also provides function *read_shape* to let us read the center point shapefile into shapely object, and plot with polygon and polylines as below:

[3]:
```
import matplotlib.pyplot as plt
from pyosp import read_shape

# Read the center point shapefile
center_shape = read_shape(center)
cx, cy = center_shape.x, center_shape.y

# Plot the swath polygon
fig, ax = plt.subplots()
swath_polygon = orig.out_polygon()
px, py = swath_polygon.exterior.xy
ax.plot(px, py)
ax.scatter(cx, cy, color='C3', label="Swath center", zorder=3)
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath polygon")
ax.legend(loc=1)

# Plot the swath profile lines
fig, ax = plt.subplots()
swath_polylines = orig.out_polylines()
for line in swath_polylines:
    x, y = line.xy
    ax.plot(x, y, color='C2')

ax.scatter(cx, cy, color='C3', label="Swath center", zorder=3)
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath profile lines")
ax.legend(loc=1)
```

[3]:
```
<matplotlib.legend.Legend at 0x7fec35b2fbe0>
```

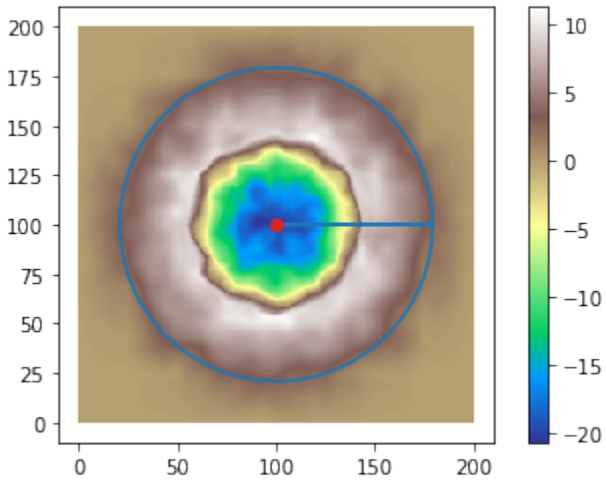




If plotting together with raster profile,

[4]:
```
import numpy as np
import gdal

ds = gdal.Open(raster)
data = ds.ReadAsArray()
gt = ds.GetGeoTransform()

xRes = gt[1]
yRes = gt[5]

# get the edge coordinates and add half the resolution
# to go to center coordinates
xmin = gt[0] + xRes * 0.5
xmax = gt[0] + (xRes * ds.RasterXSize) - xRes * 0.5
ymin = gt[3] + (yRes * ds.RasterYSize) + yRes * 0.5
ymax = gt[3] - yRes * 0.5

ds = None

# create a grid of xy coordinates
xy = np.mgrid[xmin:xmax+xRes:xRes, ymax+yRes:ymin:yRes]

fig, ax = plt.subplots()
ax.plot(px, py, color="C0", label='swath polygon')
im = ax.pcolormesh(xy[0], xy[1], data.T, cmap=plt.cm.terrain)
ax.scatter(cx, cy, color='C3', label="Swath center", zorder=3)
ax.set_aspect('equal', adjustable='box')
ax.set_xlim([-10, 210])
ax.set_ylim([-10, 210])
plt.colorbar(im)
```

[4]:
```
<matplotlib.colorbar.Colorbar at 0x7fec3410e0a0>
```



PyOSP supports exporting polygon or polylines as shapefiles. Hence the result can be read and processed in other GIS softwares like ArcGIS, QGIS, etc. For example:

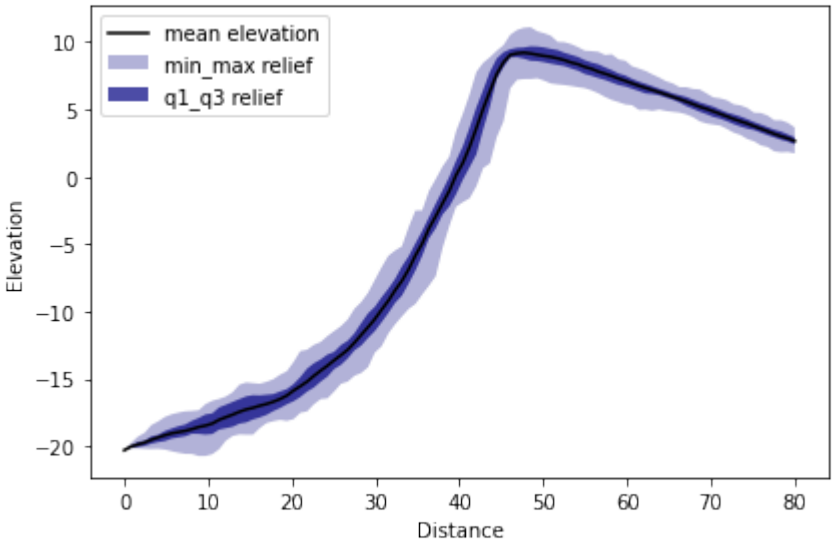
[5]:
```
# This will generate polygon and polyline shapefiles in the current folder
pyosp.write_polygon(orig.out_polygon(), "./orig_polygon.shp")
pyosp.write_polylines(orig.out_polylines(), "./orig_polylines.shp")
```

3.4 Swath profile

To plot the corresponding swath profile, simply run method:

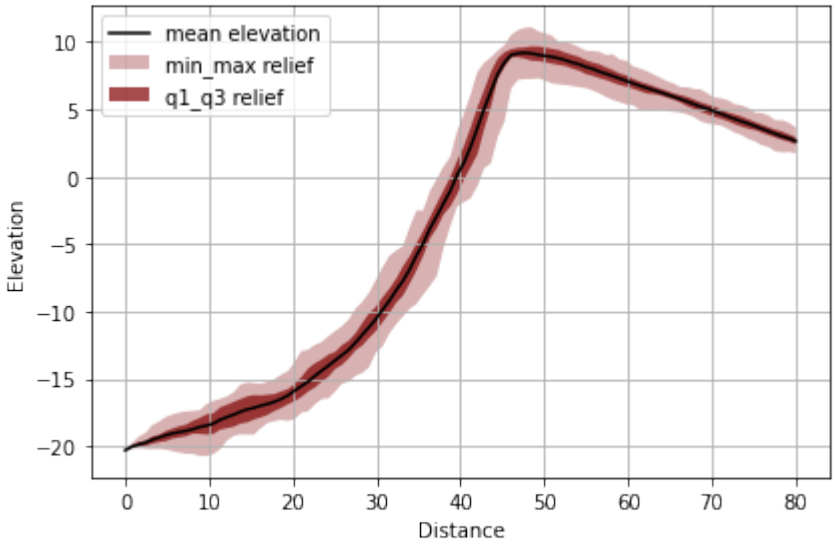
[6]:
```
orig.profile_plot()
```

[6]:
```
<AxesSubplot:xlabel='Distance', ylabel='Elevation'>
```



If figure needs to be customized, an axis argument can be passed to the object:

[7]:
```
fig, ax = plt.subplots()
orig.profile_plot(ax=ax, color="maroon")
ax.grid()
```



3.5 Summary

This section covers the basic usage of PyOSP to perform traditional fixed-radius swath profile. For more examples and advance topics, please refer to our example gallery.

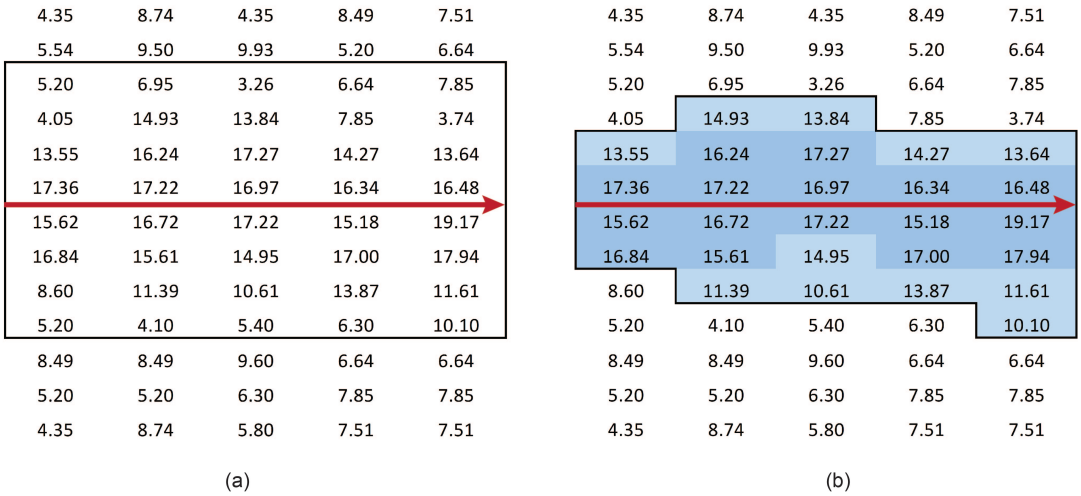
CHAPTER

FOUR

OBJECT-ORIENTED SWATH PROFILE

In last two sections we have presented using PyOSP to perform traditional fixed- width/radius analysis. Here, we give case studies to introduce proposed methods in PyOSP that can objectively characterize geological structures having complex boundaries and orientations.

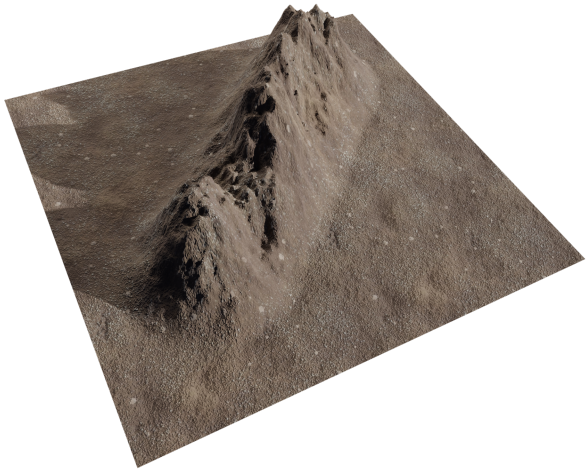
The basic concept is to combine geo-classification with swath analysis so that the landform information can be assimilated into the evaluation process. To target the specific topographic feature, PyOSP evaluates the geo-parameter values (e.g., elevation, slope, or topographic position index (TPI)) along the profile line, and truncates the line development if pixel values exceed user-defined thresholds. The process is iterated along the baseline path according to the *line_stepsize*, which enables PyOSP to collect data from non-linear profiles with irregular boundaries. Figure below illustrates the difference of sampled data between traditional (a) and proposed method (b) from data matrix perspective (min_value=10).



We use "object-oriented" to name the swath method as the key element here is to objectively define a topographic object through geo-parameter characteriation, which is in contast with previous "functional" approach that is solely parameterized through a fixed- width/radius. The comparison between these two are demonstrated through the syntheric tests as given below.

4.1 Curvilinear swath analysis

For curvilinear swath illustrative case, we use an artifacial DEM dataset that was used in the first section.



PyOSP offers elevation, slope, and topographic postion index (TPI) based swath methods in both cuvilinear (curvsp) and circular (cirsp) modules. Below gives a simple example for carrying out these geo-parameter aided analyses in curvilinear case.

[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

fixed_width = pyosp.Orig_curv(baseline, raster, width=100,
                              line_stepsize=3, cross_stepsize=None)

elev = pyosp.Elev_curv(baseline, raster, width=100,
                       min_elev=0.01,
                       line_stepsize=3, cross_stepsize=None)

slope = pyosp.Slope_curv(baseline, raster, width=100,
                         min_slope=1,
                         line_stepsize=3, cross_stepsize=None)

tpi = pyosp.Tpi_curv(baseline, raster, width=100,
                     tpi_radius=50, min_tpi=0,
                     line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

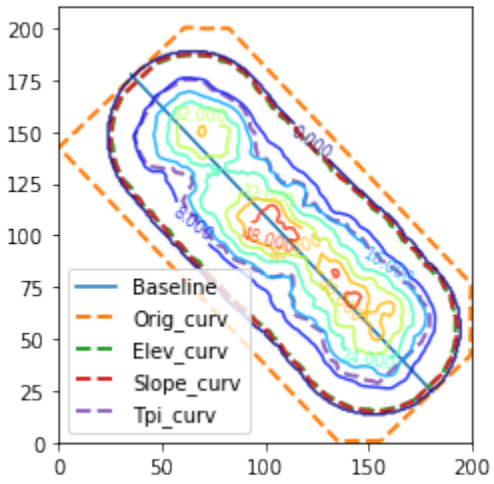
The code block above generated four swath objects corresponding to original (fixed-width), elevation, slope, and TPI methods, respectively. The default miminum and maximum value thresolds for elevation method are -inf and inf, for slope are 0 and 90, for TPI are -inf are inf. The default *line_stepsize* and *cross_stepsize* equal to the size of raster resolution. TPI based method has an additional parameter *tpi_radius* that controls the size of calculation window. For more information, users are refer to Weiss, 2001.

4.2 Swath polygons comparison

In previous section, we have shown the cases of using GDAL to plot the swath polygon on top of raster dataset. Here, we give an example by making use of rasterio to easily generate similar results. Note that PyOSP is not depend on rasterio, so the user needs to install rasterio seperately.

[2]:
```
import rasterio
from rasterio.plot import show
import matplotlib.pyplot as plt

# plot the shapely object
def plot_coords(ax, ob, **kwargs):
    x, y = ob.xy
    ax.plot(x, y, zorder=1, **kwargs)

src = rasterio.open(raster)
swath_obs = [fixed_width, elev, slope, tpi]
line_shape = pyosp.read_shape(baseline)

fig, ax = plt.subplots()
show(src.read(), transform=src.transform, ax=ax, contour=True, cmap='jet')
plot_coords(ax=ax, ob=line_shape, label="Baseline")
for ob in swath_obs:
    plot_coords(ax=ax, ob=ob.out_polygon().exterior,
                linestyle="--", lw=2, label=str(ob))

ax.set_aspect('equal', adjustable='box')
ax.legend()
```

[2]:
```
<matplotlib.legend.Legend at 0x7fe0df0726d0>
```



As shown above, both elevation and slope based methods captured the exact shape of mountain object (elevation > 0). This is achieved by setting the small minimum threshold of 0.01m and 1 degree to differentiate mountain range from surronding ground. The TPI swath goes with contour elevation=16m for a great portion, which essentially delineates the convex up (erosional) slopes from the convex down (depositional slopes) in the topography, offering some interesting insights about the topoghraphic development and could be useful in some analysis.

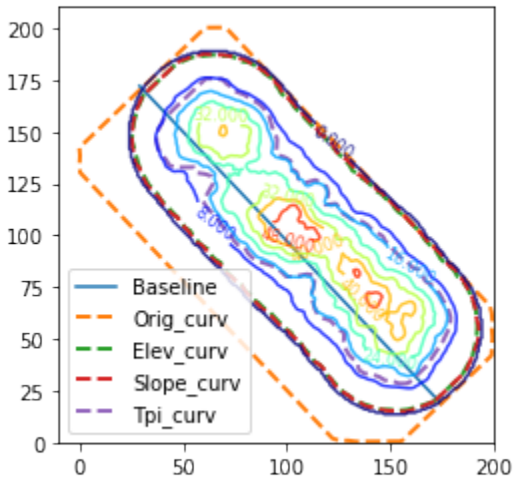
Let's do some additional sensitivity analysis about the location of baseline. This time we offset the baseline to the left

of centeral axis of the mountain. Swath analyses were applied again and generated polygons as below:

[3]:
```
baseline_offset = pyosp.datasets.get_path("homo_baselineOffset.shp") # the path to␣
↪baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

fixed_width_offset = pyosp.Orig_curv(baseline_offset, raster, width=100,
                                     line_stepsize=3, cross_stepsize=None)

elev_offset = pyosp.Elev_curv(baseline_offset, raster, width=100,
                              min_elev=0.01,
                              line_stepsize=3, cross_stepsize=None)

slope_offset = pyosp.Slope_curv(baseline_offset, raster, width=100,
                                min_slope=1,
                                line_stepsize=3, cross_stepsize=None)

tpi_offset = pyosp.Tpi_curv(baseline_offset, raster, width=100,
                            tpi_radius=50, min_tpi=0,
                            line_stepsize=3, cross_stepsize=None)

swath_offset = [fixed_width_offset, elev_offset, slope_offset, tpi_offset]
line_offset = pyosp.read_shape(baseline_offset)

fig, ax = plt.subplots()
show(src.read(), transform=src.transform, ax=ax, contour=True, cmap='jet')
plot_coords(ax=ax, ob=line_offset, label="Baseline")
for ob in swath_offset:
    plot_coords(ax=ax, ob=ob.out_polygon().exterior,
                linestyle="--", lw=2, label=str(ob))

ax.set_aspect('equal', adjustable='box')
ax.legend()
```

```
Processing: [########################] 71 of 71 lineSteps
```

[3]:
```
<matplotlib.legend.Legend at 0x7fe0defaf340>
```



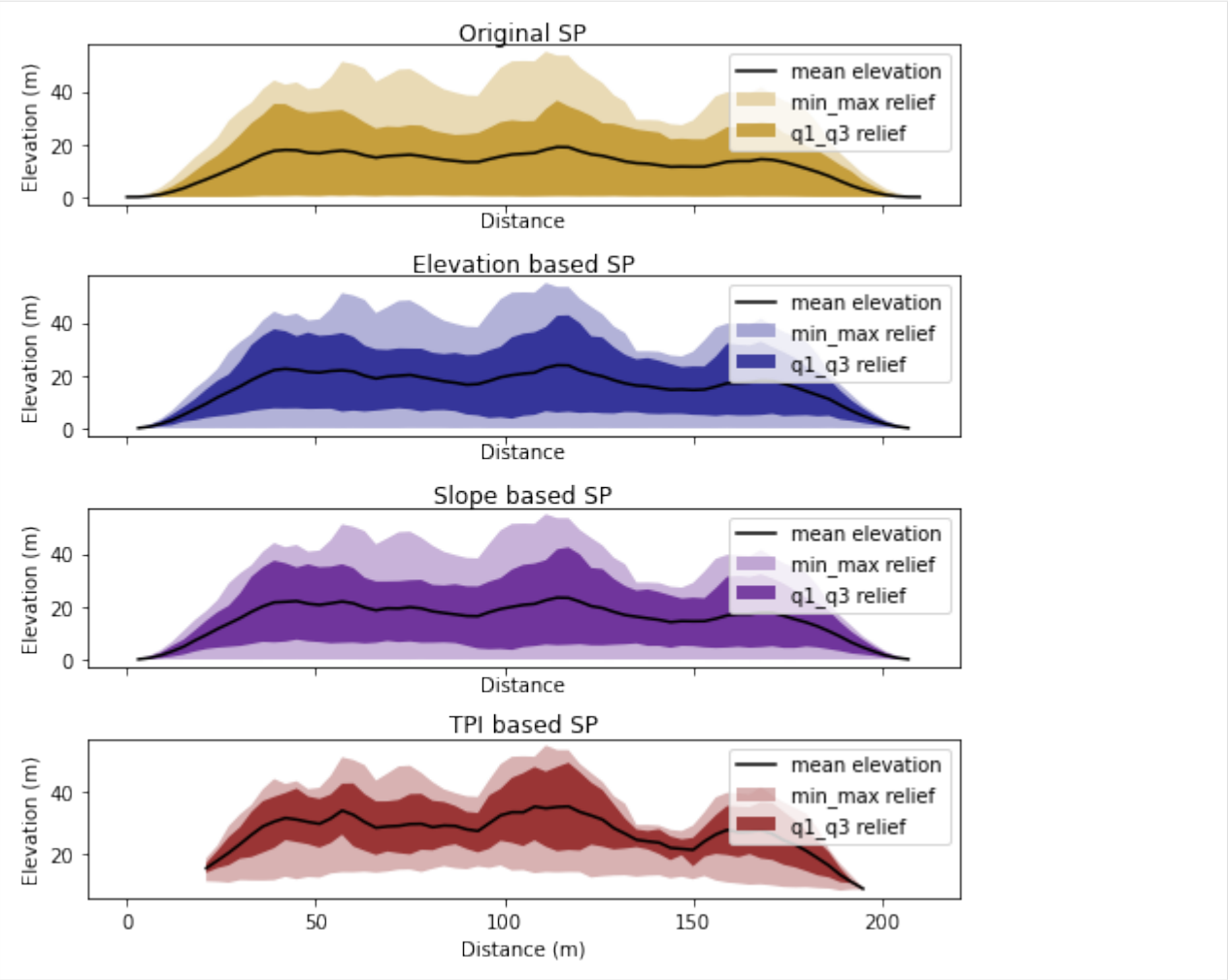
It shows that the original swath method creates a polygon offset with the altered baseline, while all geo-parameter based

swath polygons are identical to the previous case.This illustrates one of motivations behind about using object- instead of functional-oriented swath methodologies, by truncating the profile lines according to the object's geo-parameter values rather than a fixed-swath width.

4.3 Swath profile

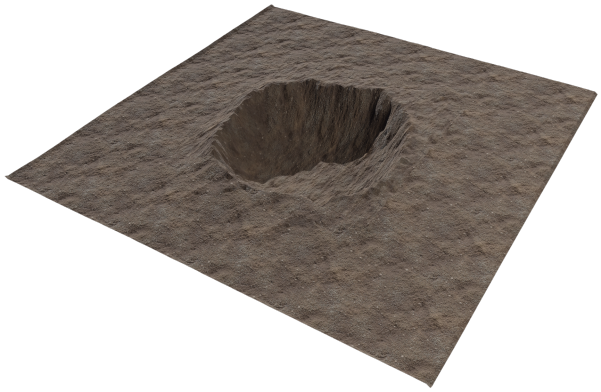
To compare the swath profiles generated from four methods, we plot them in the same figure.

[4]:
```
colors = ['darkgoldenrod', 'navy', 'indigo', 'maroon']
titles = ['Original SP', 'Elevation based SP',
          'Slope based SP', 'TPI based SP']

fig, ax = plt.subplots(4, 1, sharex=True, figsize=(7, 7))
# fig.subplots_adjust(hspace=0.2)

for ind, ob in enumerate(swath_obs):
    ob.profile_plot(ax=ax[ind], color=colors[ind])
    ax[ind].set_title(titles[ind], pad=2)
    ax[ind].set_ylabel("Elevation (m)")
    ax[ind].legend()

    if ind == len(swath_obs)-1:
        ax[ind].set_xlabel('Distance (m)')
```



The swath profiles show that the elvation mean, 3rd quartile, and the maximum elevations for the traditional, elevation based, and slope based methods are nominally similar. The lower bounds (1st quartile) for the traditional swath are overly smooth and generally match the minimum elevation line due to the inclusion of the surrounding ground surface area. The TPI based swath profiles have the same maximum elevation lines as the other methods, but the rest of the profile statistics are significantly different, due to the threshold value used only capturing the ranges upper slopes and ridges, the erosional convex-up areas.

4.4 Circular swath analysis



Synthetic topography of a depression was created to evaluate PyOSP geo-parameter based circular swath method. Note that traditional fixed-radius swath (circular) is similar to fixed-width method (curvilinear) except profile lines are following radial directions stemmed from a swath center. **PyOSP proposes a method that extend the profile line from the user-defined centroid to the highest elevation along its path, then continue beyond that elevation peak until the geo-parameter thresold or radius is reached.** As a result, only a minimum value is required for each geo-parameter based method. In the example below, the crater impacted radius is around 80m, and the parameters for different methods are listed in the table.

Swath profile method	Parameter 1	Parameter 2	Parameter 3
Original SP	radius: 80m		
Elevation based SP	radius: 80m	min_elev: 4m	
Slope based SP	radius: 80m	min_slope: 10°	
TPI based SP	radius: 80m	min_tpi: 2	tpi_radius: 50m

[5]:
```
# restart the kernel
import os
os._exit(00)
```

[1]:
```
import pyosp

center = pyosp.datasets.get_path("center.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("crater.tif")  # the path to raster file

fixed_radius = pyosp.Orig_cir(center, raster, radius=80,
                              ng_start=0, ng_end=300,
                              ng_stepsize=1, radial_stepsize=None)

elev = pyosp.Elev_cir(center, raster, radius=80,
                      min_elev=4,
                      ng_start=0, ng_end=300,
                      ng_stepsize=1, radial_stepsize=None)
```

(continues on next page)

(continued from previous page)

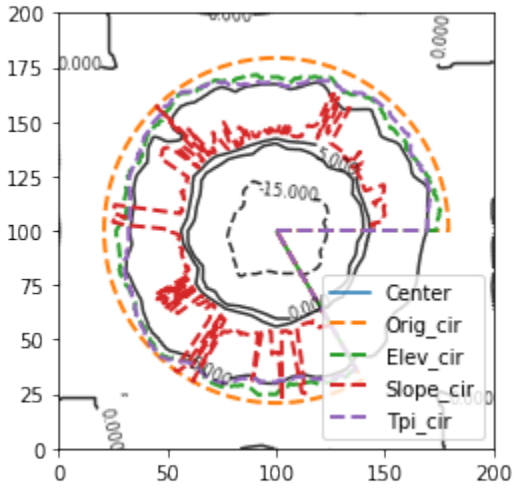
```
slope = pyosp.Slope_cir(center, raster, radius=80,
                        min_slope=10,
                        ng_start=0, ng_end=300,
                        ng_stepsize=1, radial_stepsize=None)

tpi = pyosp.Tpi_cir(center, raster, radius=80,
                    tpi_radius=50, min_tpi=2,
                    ng_start=0, ng_end=300,
                    ng_stepsize=1, radial_stepsize=None)
```

```
Processing: [#########################] 300 of 300 lineSteps
```

4.5 Swath polygons comparison

[2]:
```
import rasterio
from rasterio.plot import show
import matplotlib.pyplot as plt

# plot the shapely object
def plot_coords(ax, ob, **kwargs):
    x, y = ob.xy
    ax.plot(x, y, zorder=2, **kwargs)

src = rasterio.open(raster)
swath_obs = [fixed_radius, elev, slope, tpi]
center_shape = pyosp.read_shape(center)

fig, ax = plt.subplots()
plot_coords(ax=ax, ob=center_shape, label="Center")
for ob in swath_obs:
    plot_coords(ax=ax, ob=ob.out_polygon().exterior,
                linestyle="--", lw=2, label=str(ob))

show(src.read(), transform=src.transform, ax=ax, contour=True,
     levels=[-15, 0, 5], colors='k')
ax.set_aspect('equal', adjustable='box')
ax.legend(loc=4)
```

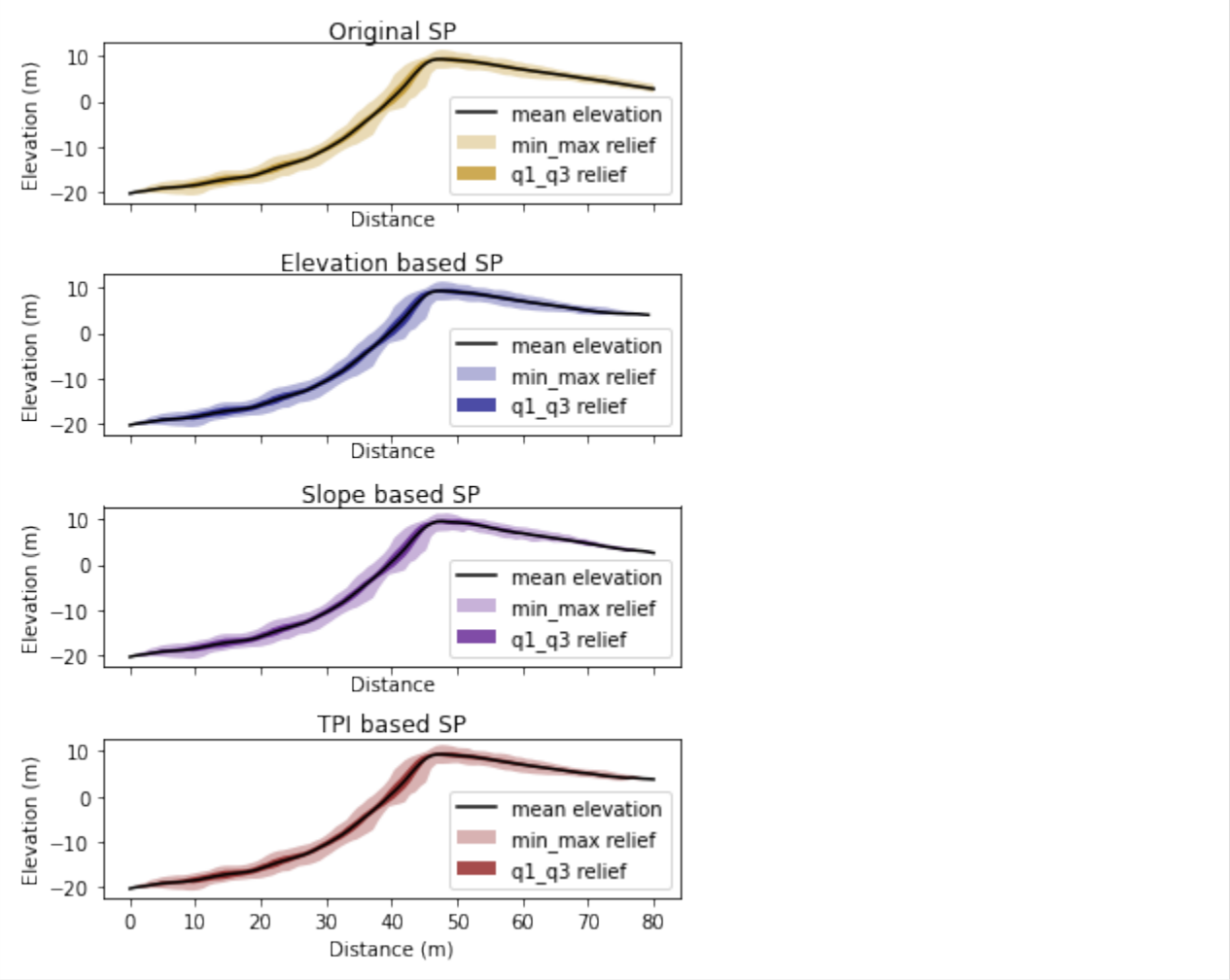
[2]:
```
<matplotlib.legend.Legend at 0x7f51c5cfb190>
```



As shown above, the traditional fixed-radius swath includes a significant proportion of irrelevant data. Elevation and TPI based methods correctly delineate the irregular boundary of the crater (contour lines of 5m), while slope based polygon is erratic with classification being limited by the sloping rim of the depression or the maximum radial length.

4.6 Swath profile

[3]:
```
colors = ['darkgoldenrod', 'navy', 'indigo', 'maroon']
titles = ['Original SP', 'Elevation based SP',
          'Slope based SP', 'TPI based SP']

fig, ax = plt.subplots(4, 1, sharex=True, figsize=(5, 7))
# fig.subplots_adjust(hspace=0.2)

for ind, ob in enumerate(swath_obs):
    ob.profile_plot(ax=ax[ind], color=colors[ind])
    ax[ind].set_title(titles[ind], pad=2)
    ax[ind].set_ylabel("Elevation (m)")
    ax[ind].legend()

    if ind == len(swath_obs)-1:
        ax[ind].set_xlabel('Distance (m)')
```



Swath profiles for all methods applied to the homogeneous crater are similar due to the circular setting of the crater and the tight bounds of the maximum allowable radius. However, for characterizing real-world complex topographic boundaries, the choice of swath method will depend on specific working condition and his/her knowledge about the studying object.

4.7 Summary

Here, we presented geo-parameter based swath analyses through the application on synthetic mountain and crater cases. The objective delineation of topographic objects increases swath profile precision; enabling reproducibility at a higher level than previous approaches. Note that PyOSP also has a rich selection of post-processing functionalities and currently supports, but is not limited to, cross swath, data categprozation and filtering, slce, histogram, and scatter swath plotting using simple and efficient workflows. For more information, the users can refer to our tutorial and library API.

CHAPTER

FIVE

CUSTOMIZED SWATH ANALYISS AND ESSENTIAL DATA STRUCTURE

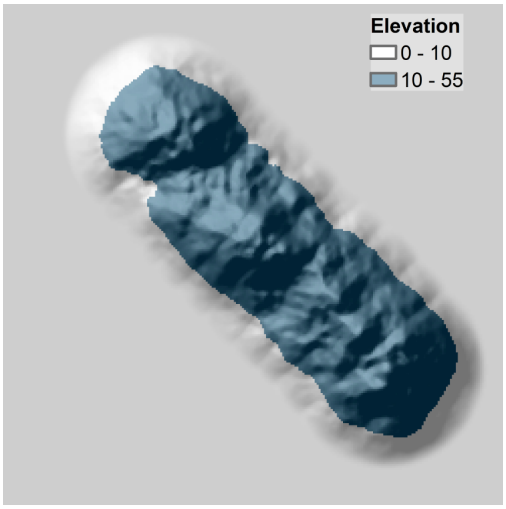
5.1 Customized swath area

PyOSP provides several geo-parameter based swath analyses, including elevation, slope, and topographic position index (TPI). However, in real applications, these options may not be sufficient for user to accurately delineate the desired topgraphic object. For example, the intended object may possess roughness>0.5, or curvture<3, among other possibilities that not being covered by PyOSP. In order to deal with this issue, we provide a tip here that allows users to perform customized swath analysis.

We use the same synthetic landscape that being used in the section *Fixed-width curvilinear swath profile*.

Here, let's suppose PyOSP does not have elevation-based characterization. We use raster analysis in other software to exact the range of elevation>10, and incoporate the resulted DEM into swath profile analysis.

5.1.1 Step 1. Raster calculation in GIS software



As shown above, we used raster calculation to extract the portion of DEM having that elevation>10.

5.1.2 Step 2. Use fixed-width swath method to generate the swath object

Now, we use fixed-width swath method to generate the swath object bounded by new DEM. As illustrated previously, PyOSP keeps detecting the boundary of DEM to truncate the analysis at the data boundary. Thus, we can set swath width beyond the possible length of profile line, so the swath area will be determined by the DEM range rather than fixed-width.

Note, PyOSP would assume no data area has cell value less then -1e+20. Make sure the raster file satisfy this criteria before the analysis.

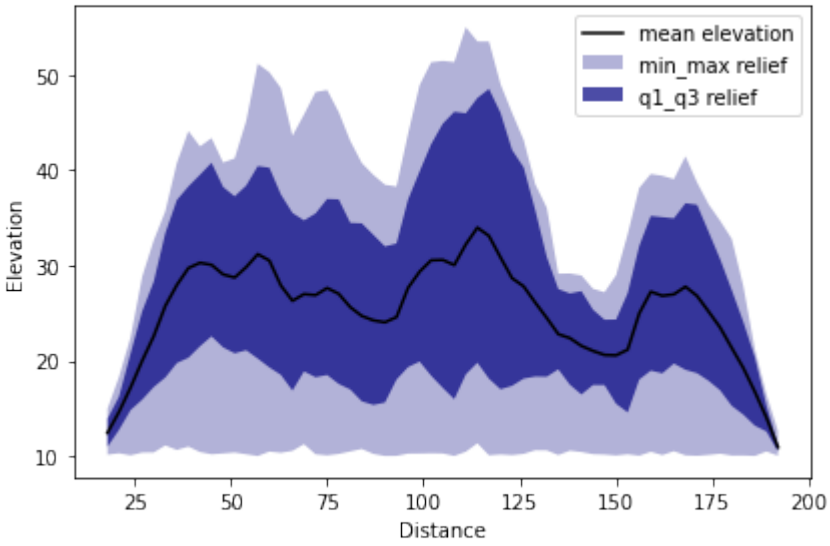
[7]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_elev10.tif")  # the path to raster file

orig = pyosp.Orig_curv(baseline, raster, width=1000,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

5.1.3 Step 3. Swath results.

Now we can verify the results by swath profile plot, or exported polygon or polylines.

[8]:
```
orig.profile_plot()
```



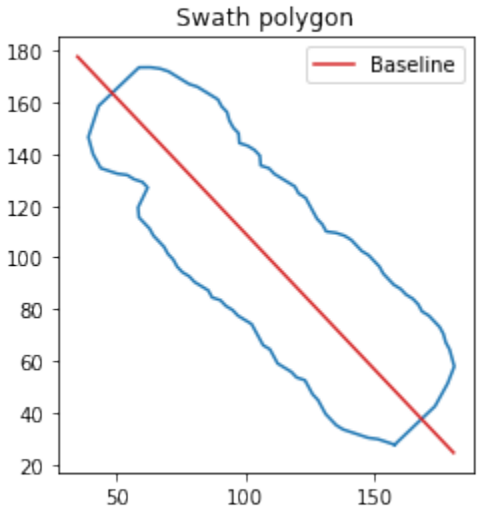
[9]:
```
import matplotlib.pyplot as plt
from pyosp import read_shape

# read the baseline shape
line_shape = read_shape(baseline)
lx, ly = line_shape.xy

# Plot the swath polygon
fig, ax = plt.subplots()
```

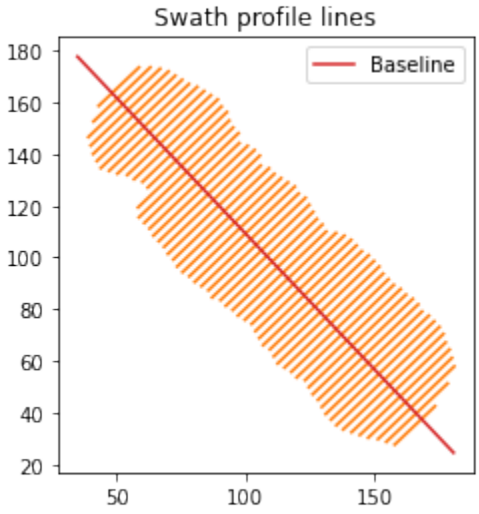
(continues on next page)

(continued from previous page)

```
swath_polygon = orig.out_polygon()
px, py = swath_polygon.exterior.xy
ax.plot(px, py)
ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath polygon")
ax.legend()

# Plot the swath profile lines
fig, ax = plt.subplots()
swath_polylines = orig.out_polylines()
for line in swath_polylines:
    x, y = line.xy
    ax.plot(x, y, color='C1')

ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath profile lines")
ax.legend()
```

[9]: <matplotlib.legend.Legend at 0x7fc020086b80>





Notice that all elevation data is above 10, and swath area is bounded by DEM data range. It demonstrates that PyOSP offers users the option to perform customized swath analysis (i.e., roughness, aspect, refief, structure, etc.). This flexibility provides the potential to incorporate any raster classification technique into the swath analysis beyond the default methods in PyOSP.

5.2 PyOSP data structure

Now, let's take a look at structure of most essential data in PyOSP.

(1) Distance

Attibute *distance* presents an array of baseline points with increasing distance from the starting point.

```
[10]: orig.distance
```

```
[10]: array([  0.,   3.,   6.,   9.,  12.,  15.,  18.,  21.,  24.,  27.,  30.,
              33.,  36.,  39.,  42.,  45.,  48.,  51.,  54.,  57.,  60.,  63.,
              66.,  69.,  72.,  75.,  78.,  81.,  84.,  87.,  90.,  93.,  96.,
              99., 102., 105., 108., 111., 114., 117., 120., 123., 126., 129.,
             132., 135., 138., 141., 144., 147., 150., 153., 156., 159., 162.,
             165., 168., 171., 174., 177., 180., 183., 186., 189., 192., 195.,
             198., 201., 204., 207., 210.])
```

(2) Lines

Lines restore coordinates of points along each profile line, from left hand side to the right. Let's examine a profile line with index of 9 (distance is 27 from the starting point).

```
[11]: orig.lines[9]
```

```
[11]: [[147.66393878283108, 30.266284945875174],
       [148.2435671297984, 30.817674988759467],
       [148.8231954767657, 31.369065031643764],
       [149.402823823733, 31.920455074528054],
       [149.98245217070033, 32.47184511741235],
       [150.56208051766765, 33.02323516029664],
```

(continues on next page)

(continued from previous page)

```
[151.14170886463495, 33.574625203180936],
[151.72133721160225, 34.12601524606523],
[152.30096555856957, 34.67740528894952],
[152.88059390553687, 35.228795331833815],
[153.4602222525042, 35.78018537471811],
[154.0398505994715, 36.3315754176024],
[154.61947894643882, 36.882965460486695],
[155.19910729340612, 37.43435550337099],
[155.77873564037344, 37.98574554625528],
[156.35836398734074, 38.537135589139574],
[156.93799233430806, 39.088525632023874],
[157.51762068127536, 39.63991567490817],
[158.0972490282427, 40.19130571779246],
[158.67687737520998, 40.74269576067675],
[159.2565057221773, 41.294085803561046],
[159.8361340691446, 41.84547584644534],
[160.41576241611193, 42.39686588932963],
[160.99539076307923, 42.948255932213925],
[161.57501911004655, 43.49964597509822],
(162.15464745701385, 44.05103601798251),
[162.73427580398115, 44.602426060866804],
[163.31390415094847, 45.1538161037511],
[163.89353249791577, 45.70520614663539],
[164.4731608448831, 46.25659618951968],
[165.0527891918504, 46.80798623240398],
[165.63241753881772, 47.35937627528827],
[166.21204588578502, 47.91076631817256],
[166.79167423275234, 48.462156361056856],
[167.37130257971964, 49.01354640394115],
[167.95093092668696, 49.56493644682545],
[168.53055927365426, 50.11632648970974],
[169.1101876206216, 50.667716532594035],
[169.68981596758888, 51.21910657547833],
[170.2694443145562, 51.77049661836262],
[170.8490726615235, 52.321886661246914],
[171.42870100849083, 52.87327670413121],
[172.00832935545813, 53.4246667470155],
[172.58795770242546, 53.97605678989979],
[173.16758604939275, 54.527446832784086],
[173.74721439636005, 55.07883687566838],
[174.32684274332738, 55.63022691855267],
[174.9064710902947, 56.18161696143697],
[175.486099437262, 56.73300700432126],
[176.0657277842293, 57.28439704720556],
[176.64535613119662, 57.835787090089850],
[177.22498447816392, 58.387177132974145],
[177.80461282513124, 58.93856717585844],
[178.38424117209854, 59.48995721874273],
[178.96386951906587, 60.041347261627024],
[179.54349786603316, 60.59273730451132],
[180.1231262130005, 61.14412734739561]]
```

(3) data

Attibute *data* provides raster values along each profile line. For example, values along the same profile line as presented above.

[12]:
```
orig.dat[9]
```

[12]:
```
[10.366577,
 10.775647,
 10.985182,
 11.849683,
 12.324719,
 12.434934,
 13.207694,
 14.339156,
 14.339156,
 15.918287,
 17.868557,
 18.705082,
 20.37459,
 22.935509,
 23.32269,
 25.494387,
 27.356518,
 28.160154,
 27.377426,
 28.819553,
 27.965193,
 26.129932,
 26.043837,
 26.854143,
 26.317019,
 26.637743,
 27.222136,
 26.62463,
 26.012062,
 24.822708,
 25.181057,
 23.875961,
 22.917784,
 23.280067,
 21.87542,
 20.613186,
 19.87687,
 19.87687,
 19.525146,
 19.106987,
 17.9117,
 18.822304,
 19.342886,
 18.516066,
 19.936953,
 20.327038,
 20.0634,
 18.608377,
 19.296824,
```

(continues on next page)

(continued from previous page)

```
17.993258,
16.700563,
15.849781,
16.132235,
14.963792,
13.446796,
13.099882,
11.64522]
```

5.3 Summary

We introduced herein using customized option to perform swath analysis beyond default methods in PyOSP. The afforded flexibility provides the potential to incorporate any raster classification technique into swath analysis. In addition, we walked through essential data attributes of PyOSP swath object, users should be able to further analysis, plot, or conduct other post-processings by making use of numpy, Matplotlib, SciPy, among other Python packages.

CHAPTER

SIX

SLICE AND HISTOGRAM

Slice and histogram can be useful in some studies such as river terrace identification, or asymmetrical development of landscape. Here, we introduce the methods to plot slice and histogram under curvilinear and circular conditions.

6.1 Curvilinear case

6.1.1 Histogram plot

We use the same synthetic landscape that being used in the section *Fixed-width curvilinear swath profile*.

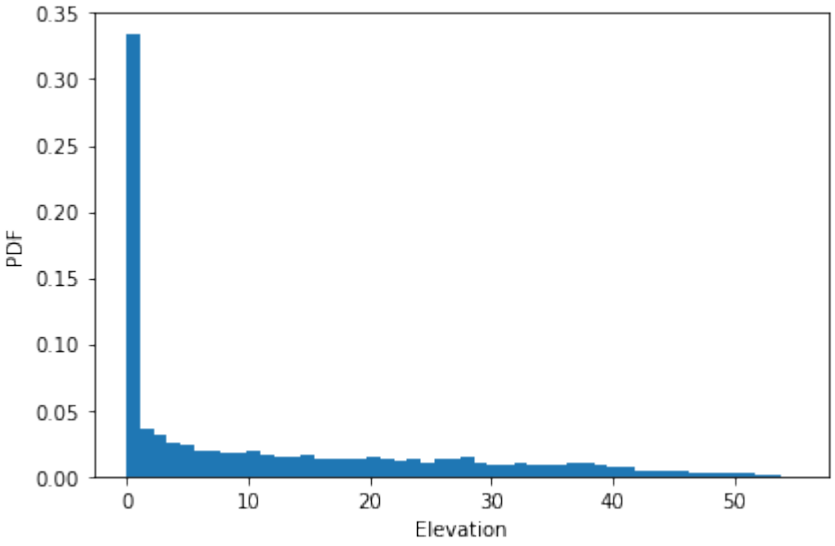
[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

orig = pyosp.Orig_curv(baseline, raster, width=100,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

We can plot histogram of all swath data.

[2]:
```
orig.hist()
```

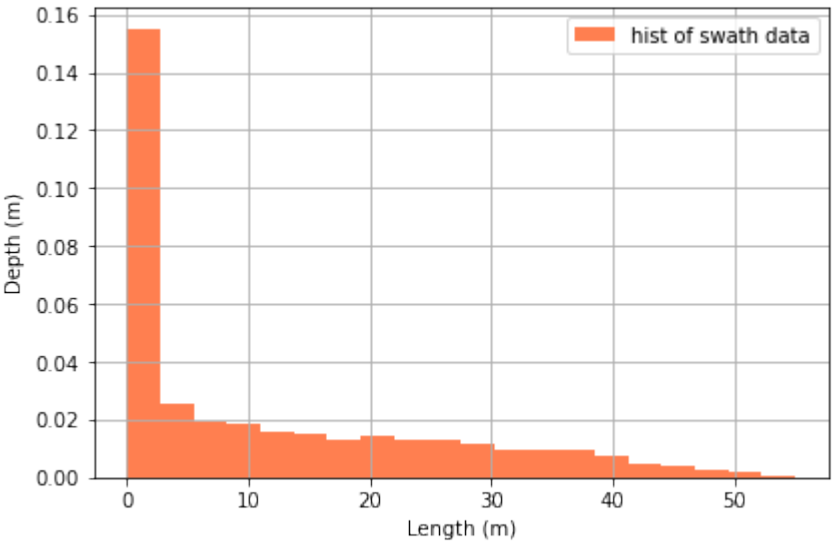
[2]:
```
<AxesSubplot:xlabel='Elevation', ylabel='PDF'>
```



We can make some changes to the plot by passing an ax object to the method.

[3]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
orig.hist(ax=ax, bins=20, color="coral", label="hist of swath data")
ax.set_xlabel("Length (m)")
ax.set_ylabel("Depth (m)")
ax.grid()
ax.legend()
```

[3]:
```
<matplotlib.legend.Legend at 0x7fef680d5dc0>
```

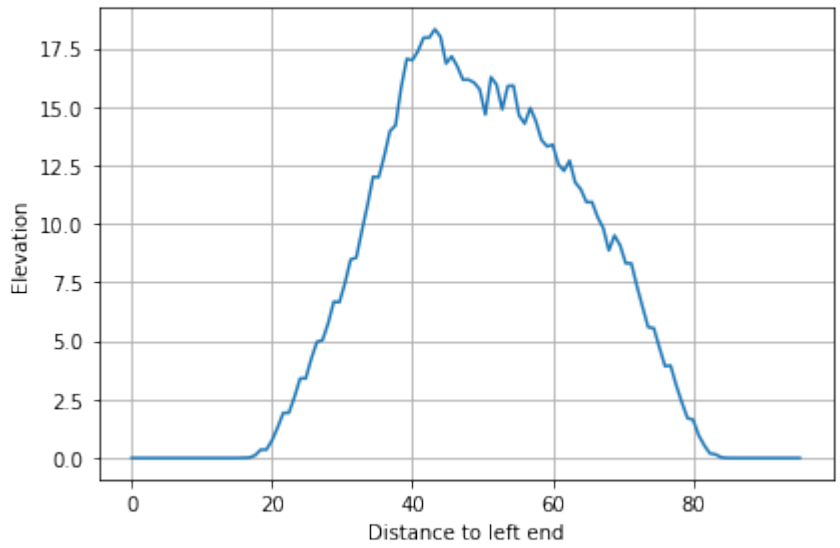


6.1.2 Slice plot

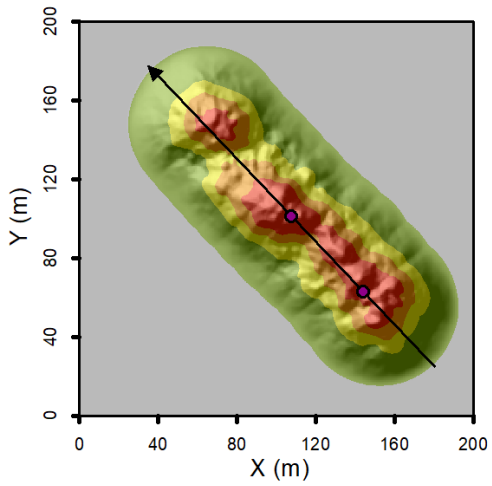
For plotting the cross-section, PyOSP provides two ways to determine the location of a slice. In the first method, user can pass a value indicating the slice distance from the starting point:

[4]:
```
orig.slice_plot(loc=20)
```

[4]:
```
<AxesSubplot:xlabel='Distance to left end', ylabel='Elevation'>
```



However, it is inconvenient in some cases by using distance to locate a cross-section. PyOSP provides an alternative method. The user can draw points on the baseline to indicate the location of the desired slice, and pass these points information to PyOSP. As shown below, we have two points drew on the map, now we plot the corresponding slices.

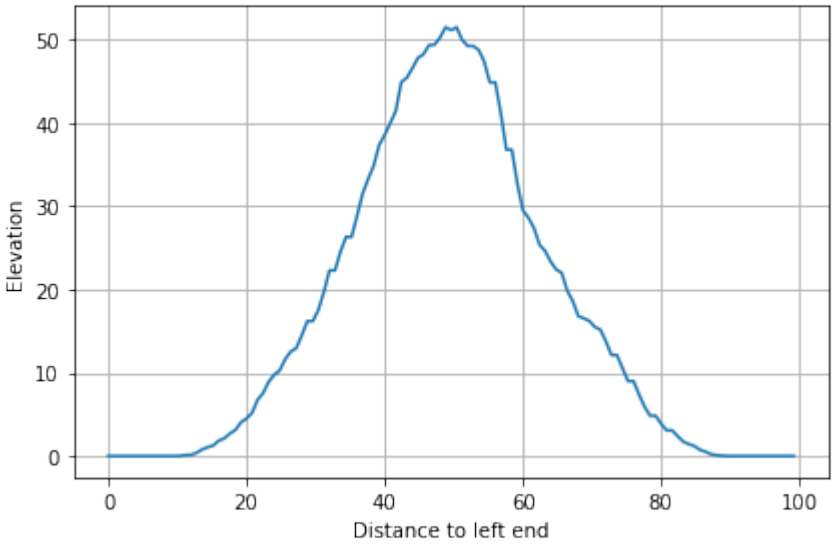


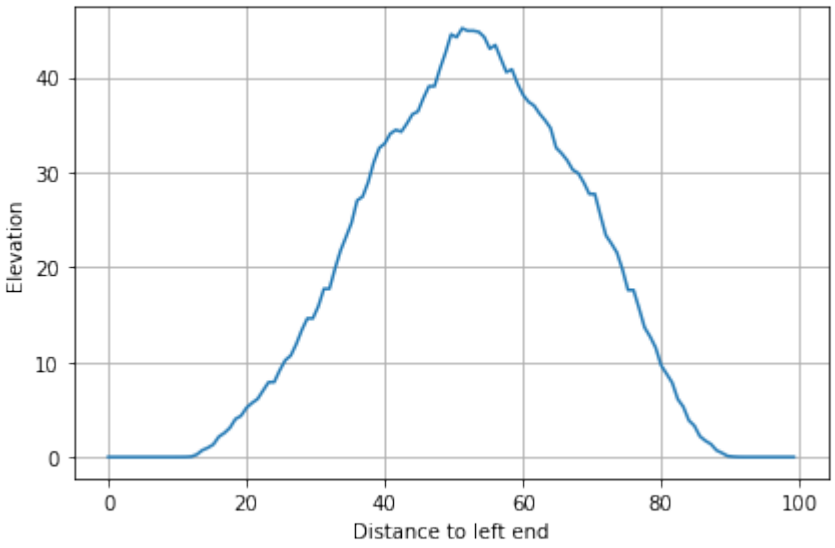
[5]:
```
from pyosp import point_coords

pointsPath = pyosp.datasets.get_path("homo_start_end.shp") # the path to the points

pointsCoords = point_coords(pointsPath)
```

(continues on next page)

(continued from previous page)

```
for point in pointsCoords:
    orig.slice_plot(loc=point)
```

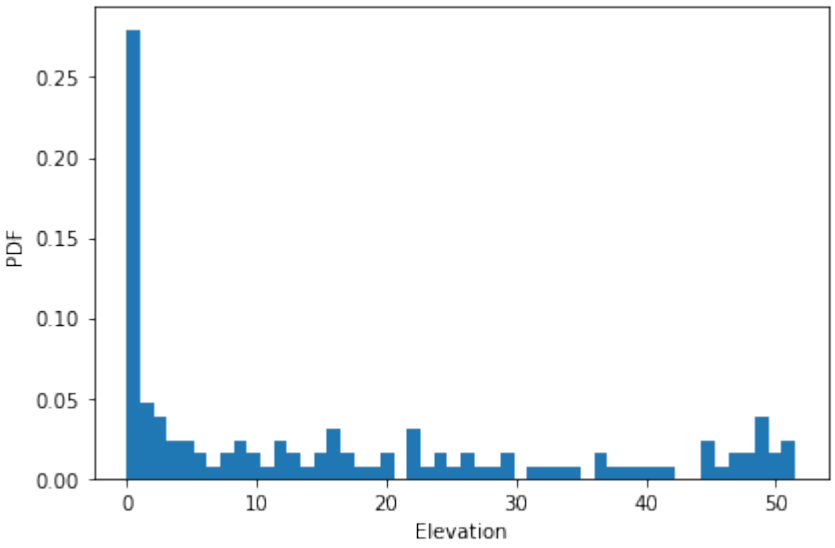


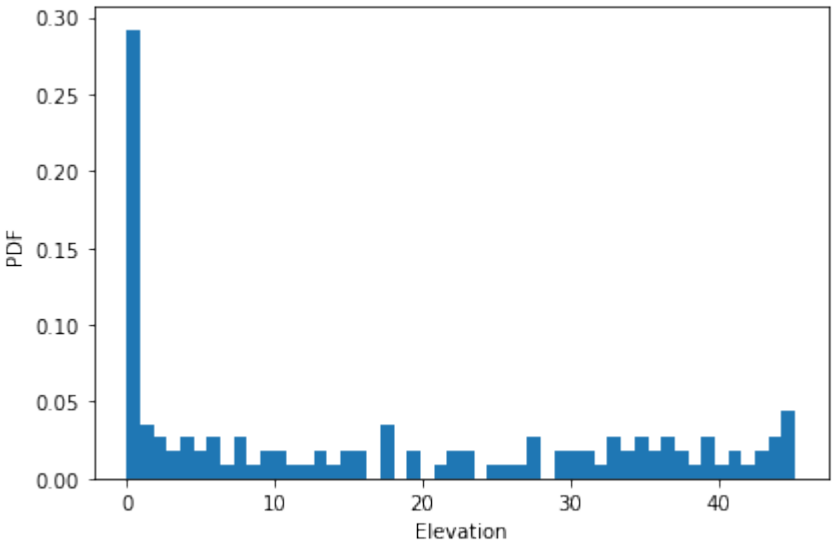


6.1.3 Histogram of slice

Further, we can plot histogram of slice data.

[6]:
```
for point in pointsCoords:
    orig.slice_hist(loc=point)
```





6.2 Circular case

6.2.1 Histogram plot

We use the same synthetic landscape that being used in the section *Fixed-radius circular swath profile*.

[ ]:
```
# restart the kernel
import os
os._exit(00)
```

[1]:
```
import pyosp

center = pyosp.datasets.get_path("center.shp")  # path to the central point shapefile
```

(continues on next page)

(continued from previous page)

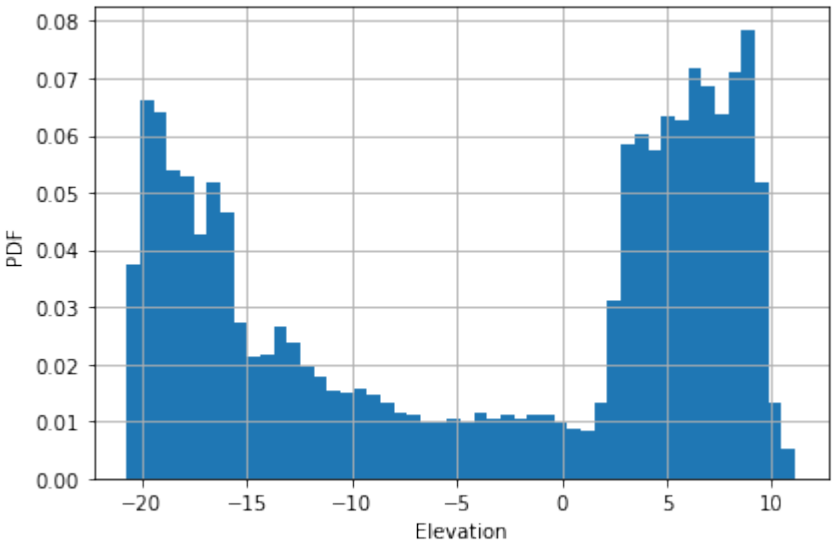
```
raster = pyosp.datasets.get_path("crater.tif")  # path to the raster tif

orig = pyosp.Orig_cir(center, raster, radius=80,
                      ng_start=0, ng_end=360,
                      ng_stepsize=5, radial_stepsize=None)
```

```
Processing: [########################] 72 of 72 lineSteps
```

Plot the histogram for all swath data

[2]:
```
orig.hist()
```

[2]:
```
<AxesSubplot:xlabel='Elevation', ylabel='PDF'>
```

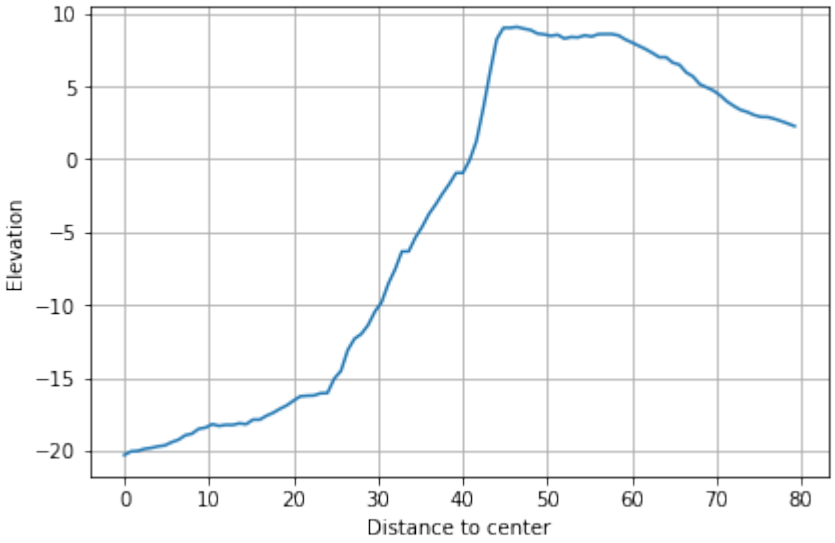


6.2.2 Slice plot

To plot a slice of circular profile, user needs to pass the *angle* that is referenced to the horizontal line.

[3]:
```
orig.slice_plot(angle=30)
```

[3]:
```
<AxesSubplot:xlabel='Distance to center', ylabel='Elevation'>
```

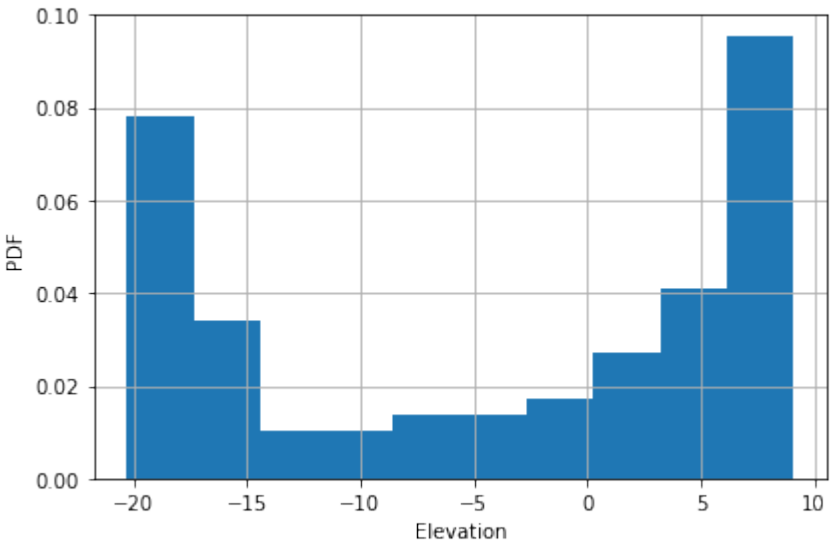


6.2.3 Slice histogram

PyOSP also provides histogram of slice in circular condition.

[4]:
```
orig.slice_hist(angle=30, bins=10)
```

[4]:
```
<AxesSubplot:xlabel='Elevation', ylabel='PDF'>
```



6.3 Summary

In this section, we introduced the methods to plot the histogram and slice for curvilinear and circular swath cases. The combination of these methods with object-oriented swath analysis can provide useful insights in terms of cross-section and data distribution of the studying object.

CHAPTER

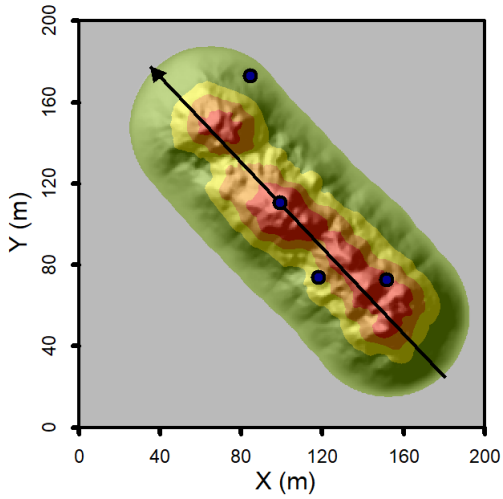
SEVEN

SWATH PROFILE WITH SCATTER PLOT

It is sometimes desired to plot the geological features with the swath profile. For example, plot the glacial points coupling with the mountain range profile (Dortch et al. 2011). PyOSP provides simple workflow to plot such figures.

7.1 Step 1. Save features as points in shapefile

As shown below, we randomly drew four points within the range of mountain. These are saved as *checking_points.shp*.



7.2 Step 2. Generate the swath object

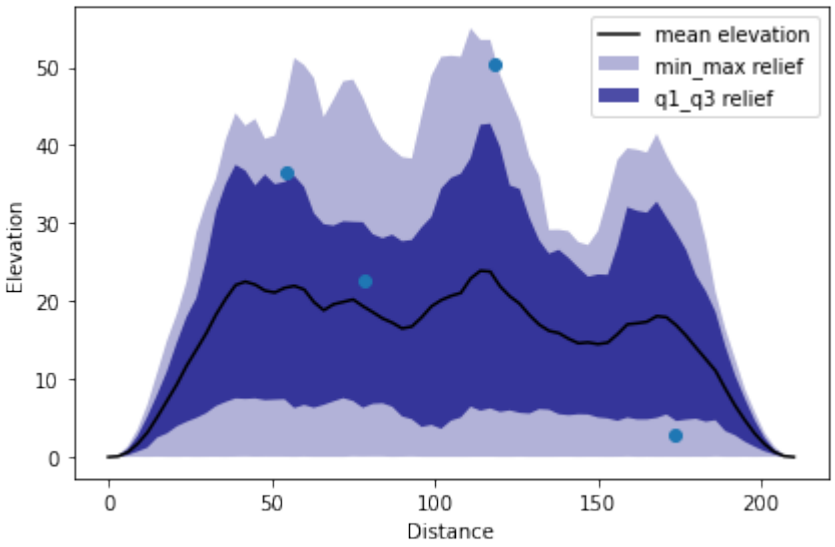
[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

elev = pyosp.Elev_curv(baseline, raster, width=100,
                       min_elev=0.01,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

7.3 Step 3. Plot the scatter with the swath profile

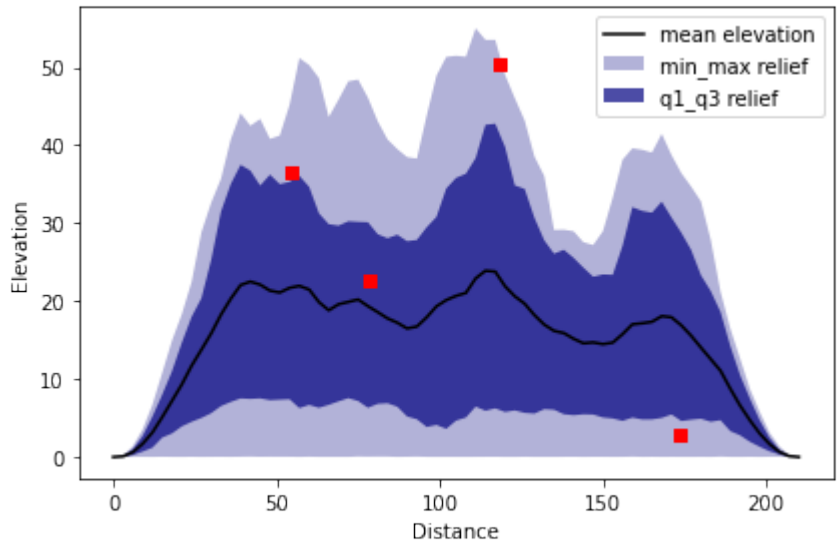
If we pass the points path to the method *profile_plot*, these points' distance and elevation information will be processed and plot in the same figure of swath profile.

[2]:
```
from pyosp import point_coords

pointsPath = pyosp.datasets.get_path("checking_points.shp")

elev.profile_plot(points=pointsPath)
```



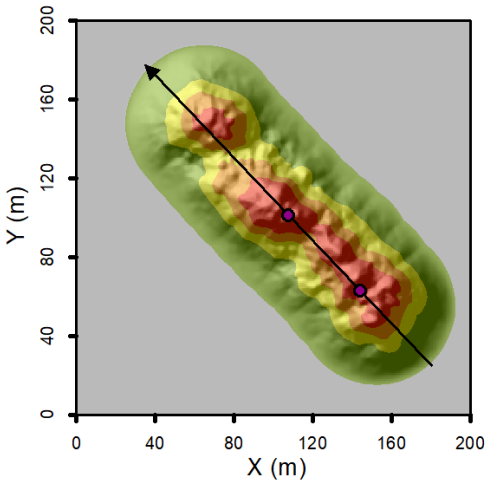
We can make some changes to the figure to make it clear.

[3]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
# Note parameter "color" means color of swath profile, "c" and additional
# parameters are passed to the Matplolib scatter function.
elev.profile_plot(ax=ax, points=pointsPath, color="navy", c="red", marker="s")
```

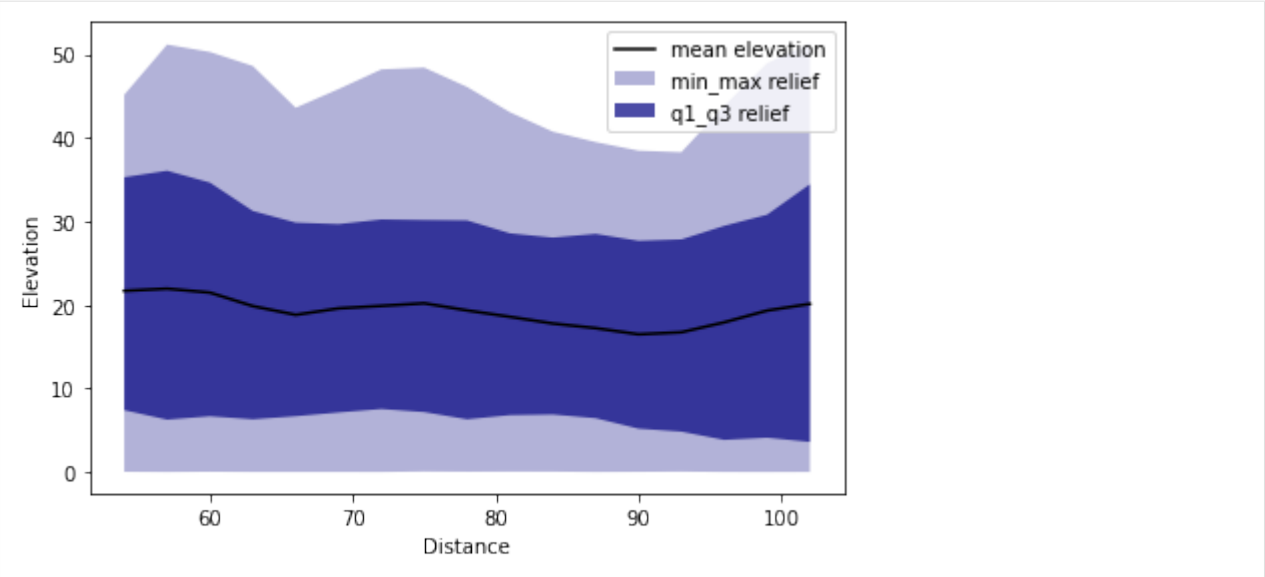


Note that method *profile_plot* has addotional parameters *start* and *end*. By defining those, user can limit the distance range to be plotted.

User can offer simple values to these parameters. Also, parameters can be defined by points on baseline to indicate the starting and ending locations, as shown below.



[5]:
```
from pyosp import point_coords

pointsPath = pyosp.datasets.get_path("homo_start_end.shp") # the path to the points

pointsCoords = point_coords(pointsPath)

# we drew the ending points first here
elev.profile_plot(start=pointsCoords[1], end=pointsCoords[0])
```


mean elevation
min_max relief
q1_q3 relief
Elevation
Distance
0
10
20
30
40
50
60
70
80
90
100

CHAPTER

EIGHT

CROSS-SWATH ANALYSIS

Cross-swath profile refers to swath analysis along tranverse rather than longitudinal direction. It has been demonstrated useful in many previous studies (e.g., Foster et al. 2010 or Dortch et al. 2011).

However, these research have restricted the cross-swath inside of a rectangular box, which without reference to geomorphic irregular boudary and are oftentimes subjective about their lengths and orientations.

To address these shorcomings, PyOSP provides more powerful and objective method to perform cross-swath analysis.

8.1 Step 1. Generate a swath objective

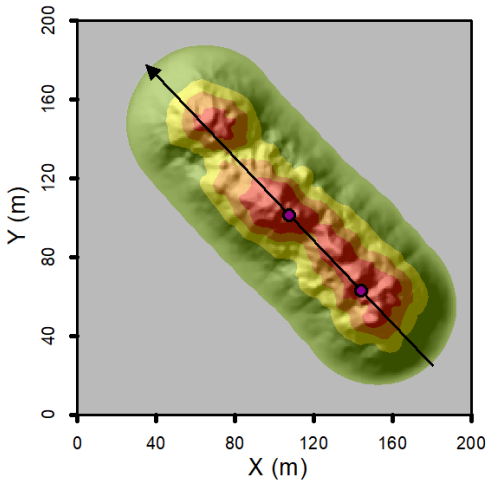
We use the same synthetic landscape that being used in the section *Fixed-width curvilinear swath profile*.

[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

elev = pyosp.Elev_curv(baseline, raster, width=100,
                       min_elev=0.01,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

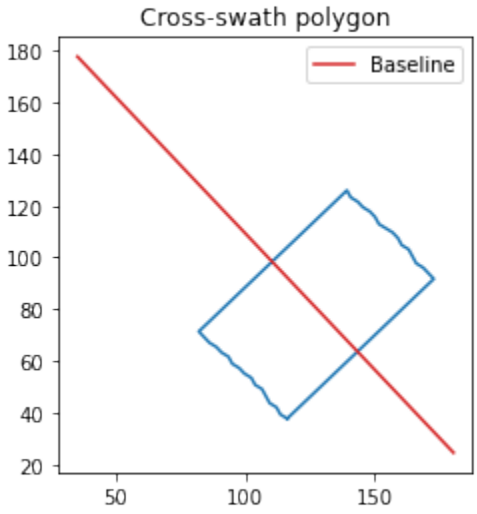
8.2 Step 2. Define the starting and ending location of cross-swath

As we demonstrated in previous section, the starting and ending locations can be either defined by inputing values indicating distances from the starting point of baseline, or using points drawn on the baseline to specify the starting and ending locations. Here, we use the second method to bound the area.



[2]:
```
import matplotlib.pyplot as plt
from pyosp import point_coords, read_shape

pointsPath = pyosp.datasets.get_path("homo_start_end.shp") # the path to the points

pointsCoords = point_coords(pointsPath)

# The sequence of the points follows drawing order on the map
# In this case, we drew the ending point first.
starting = pointsCoords[1]
ending = pointsCoords[0]

# let's verify the locations by polygon (optional)
# read the baseline shape
line_shape = read_shape(baseline)
lx, ly = line_shape.xy

fig, ax = plt.subplots()
swath_polygon = elev.out_polygon(start=starting, end=ending)
px, py = swath_polygon.exterior.xy
ax.plot(px, py)
ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Cross-swath polygon")
ax.legend()
```

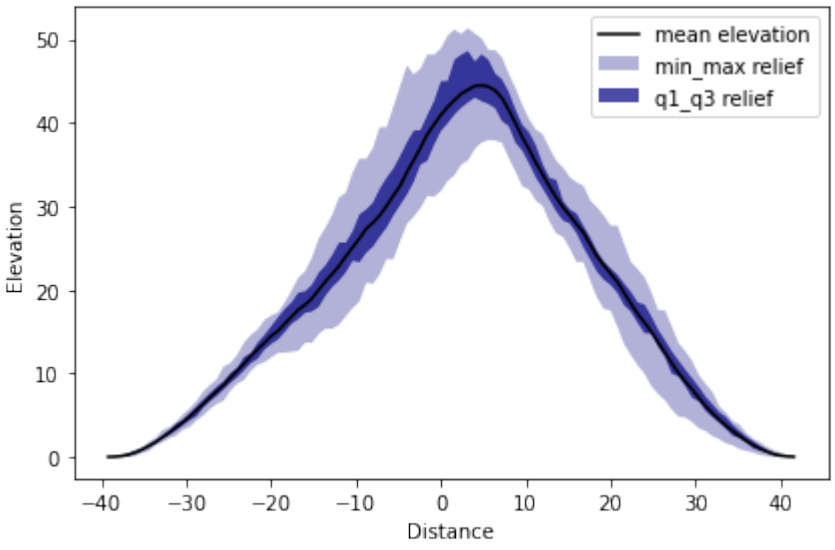
[2]:
```
<matplotlib.legend.Legend at 0x7f1681991520>
```



8.3 Step 3. Plot the cross-swath profile

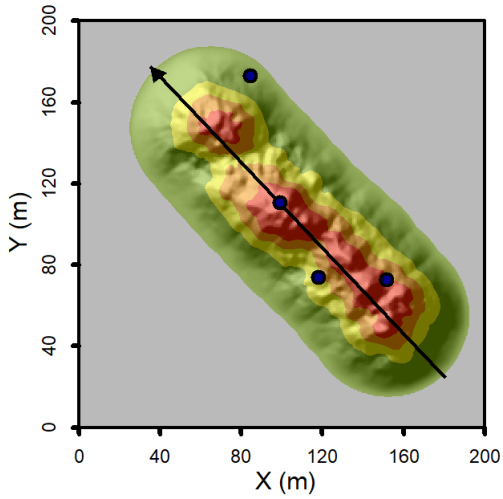
The cross-swath profile can be easily plotted by method *cross_plot*. The distance along x-axis means the transvers distance from the baseline.

[3]:
```
elev.cross_plot(start=starting, end=ending)
```

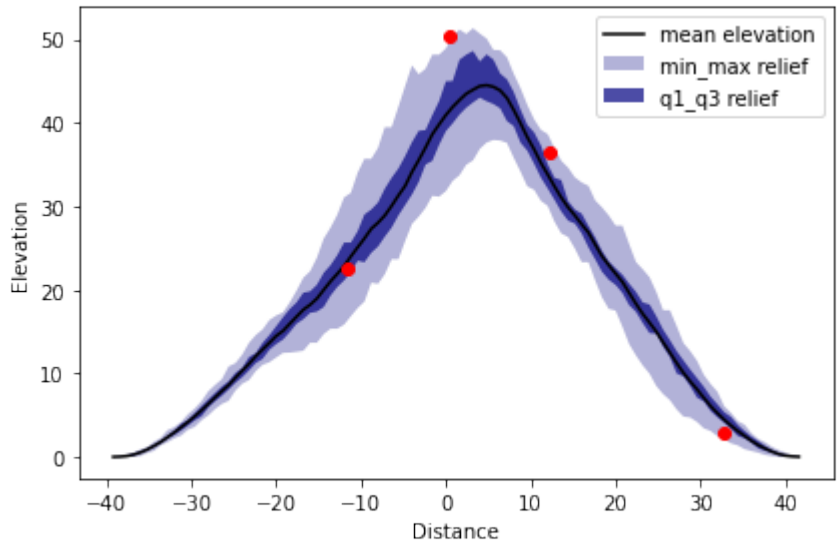


8.4 Step 4. Plot scatters with cross-swath profile (optional)

Note we can also plot scatters with cross-swath, just serve the path of points shapefile to the *cross_plot* method



[4]:
```
checkingPoints = pyosp.datasets.get_path("checking_points.shp") # the path to the points

fig, ax = plt.subplots()
# Note parameter "color" means color of swath profile, "c" and additional
# parameters are passed to the Matplolib scatter function.
elev.cross_plot(ax=ax, start=starting, end=ending, points=checkingPoints, color="navy",␣
↪c="red")
```



Noticing that some points are out of range of cross-swath along longitudinal axis, but they are still projected to the swath profile. If it is not desired, user should delete all unwanted points before the analysis.

CHAPTER

NINE

DENSITY(HEAT) SCATTER PLOT

PyOSP supports density scatter plot of swath analysis. Unlike swath profile, density scatter plot shows every elevation data on the figure, with heatmap-like coloring of the markers.

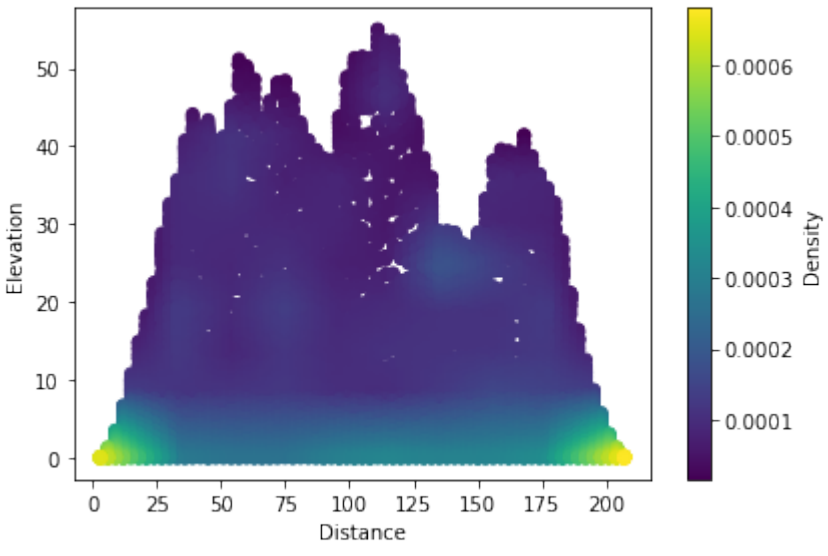
9.1 Along longitudinal direction

[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

elev = pyosp.Elev_curv(baseline, raster, width=100,
                       min_elev=0.01,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [#########################] 71 of 71 lineSteps
```

[2]:
```
elev.density_scatter()
```

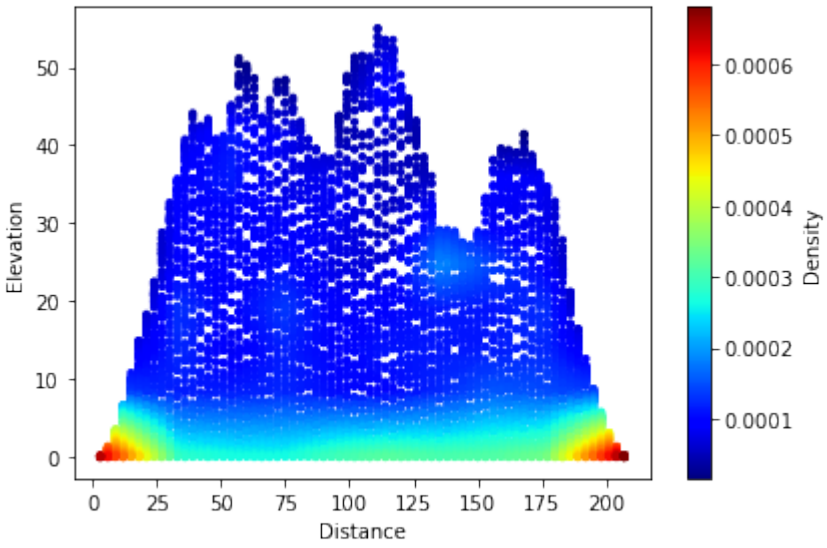
[2]:
```
<AxesSubplot:xlabel='Distance', ylabel='Elevation'>
```



We can choose other colormaps, or parameters regarding Matplotlib scatter plot.

[3]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
elev.density_scatter(ax=ax, bins=10, cmap="jet", s=10)
```

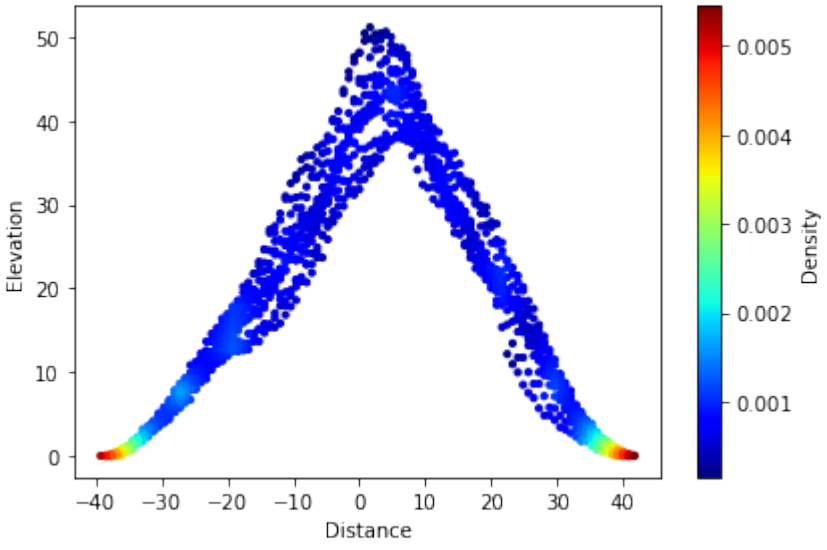
[3]:
```
<AxesSubplot:xlabel='Distance', ylabel='Elevation'>
```



9.2 Along cross-section

Density scatter can be also plotted for cross-swath analysis.

[4]:
```
fig, ax = plt.subplots()
elev.cross_plot(density_scatter=True, ax=ax, start=50, end=100, cmap="jet", s=10)
```



CHAPTER

TEN

DATA RECLASSIFICATION

Data reclassification of swath profile may not be dicusses in the current literature. However, this offered functionality by PyOSP could be of interest to many geomorphologist. For example, in a characterized swath profile, what is the statistics to which having TPI>0? Or, TPI<0?

PyOSP provides post-process of swath data according to elevation, slope, or TPI classification. Below is an example of TPI reclassificationo of elevation-based swath object.

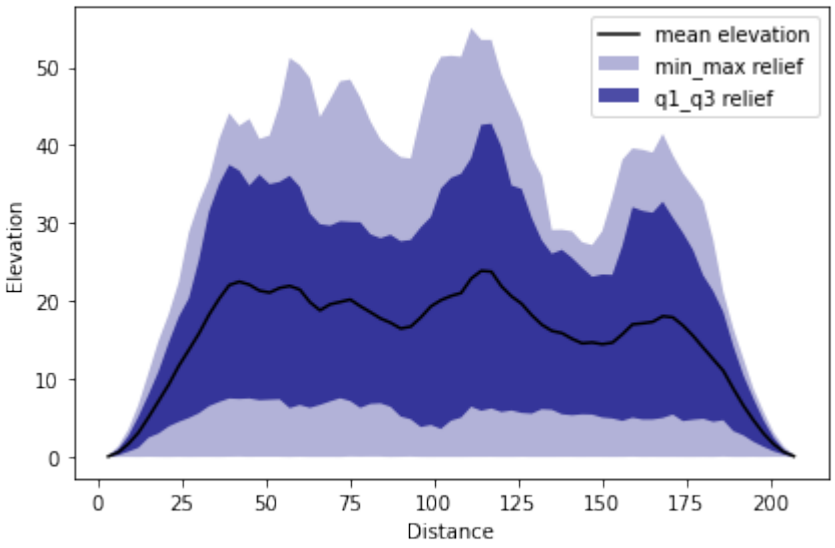
10.1 Step 1. Generate an elevation-based swath object

[1]:
```
import pyosp

baseline = pyosp.datasets.get_path("homo_baseline.shp") # the path to baseline shapefile
raster = pyosp.datasets.get_path("homo_mount.tif")  # the path to raster file

elev = pyosp.Elev_curv(baseline, raster, width=100,
                       min_elev=0.01,
                       line_stepsize=3, cross_stepsize=None)
```

```
Processing: [########################] 71 of 71 lineSteps
```

Now, let's plot the corresponding swath profile.

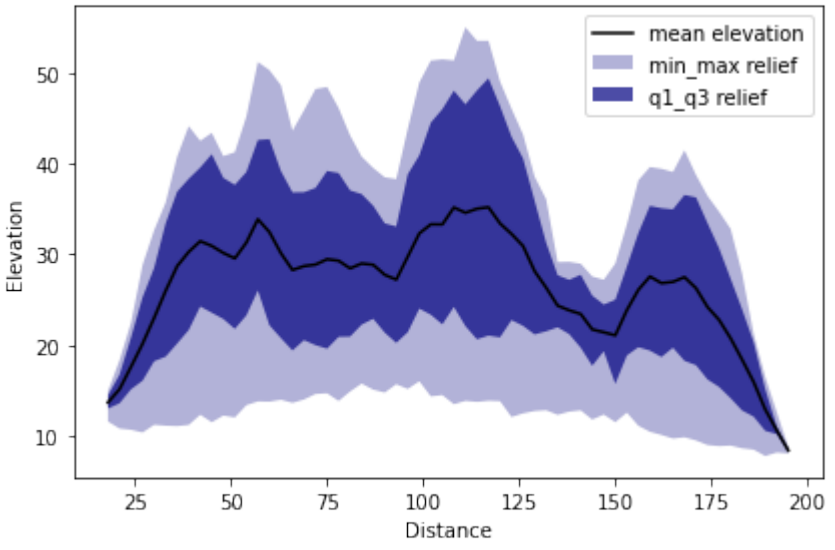
[2]:
```
elev.profile_plot()
```



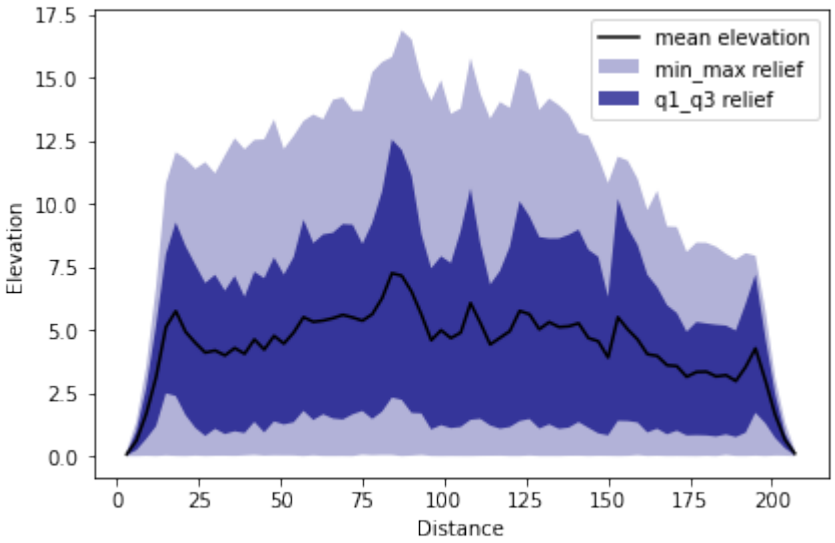
10.2 Step 2. Reclassification with swath data (TPI in this case)

In addition to what plotted above, we want to further explore, for example, the swath profile with data points having TPI>0 and TPI<0, respectively. Since TPI = 0 relates to the inflection of slope shape where topography changes from concave-down (TPI > 0) to convex up (TPI < 0). Thus, these two swaths enable the separation of topography via geomorphic processes, with positive and negative TPI representing erosional and depositional areas of the topography, respectively.

[3]:
```
# radius refers to TPI radius
# swath profile with points having TPI>0
dat = elev.post_tpi(radius=50, min_val=0, swath_plot=True)
```



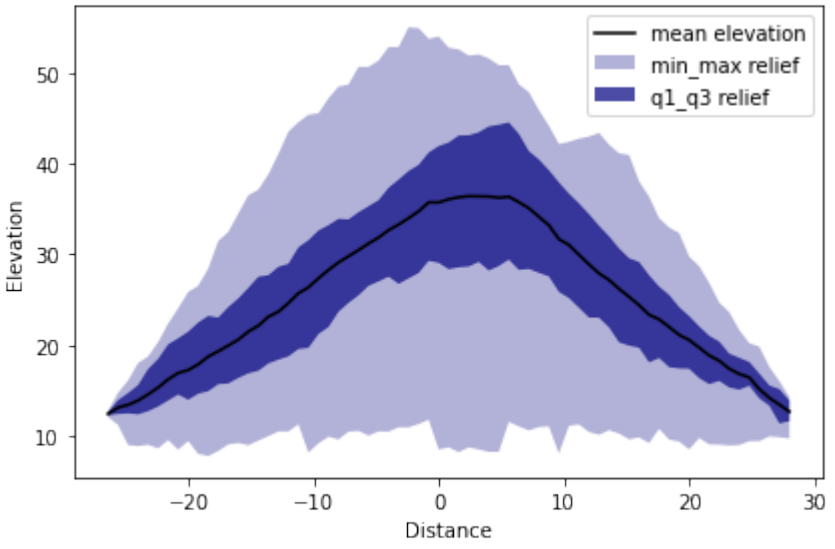
[4]:
```
# swath profile with points having TPI<0
dat = elev.post_tpi(radius=50, max_val=0, swath_plot=True)
```



10.3 Step 3. Reclassification with cross-swath analysis (optional)

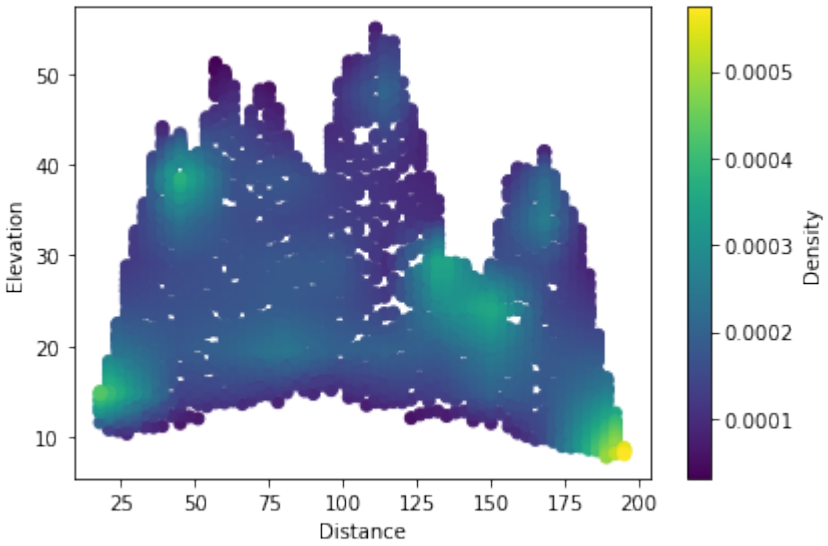
Post-processing can also be coupled with cross-swath analysis. For example, we can plot the cross-swath with reclassification of TPI>0.

[5]:
```
dat = elev.post_tpi(radius=50, min_val=0, swath_plot=True, cross=True)
```



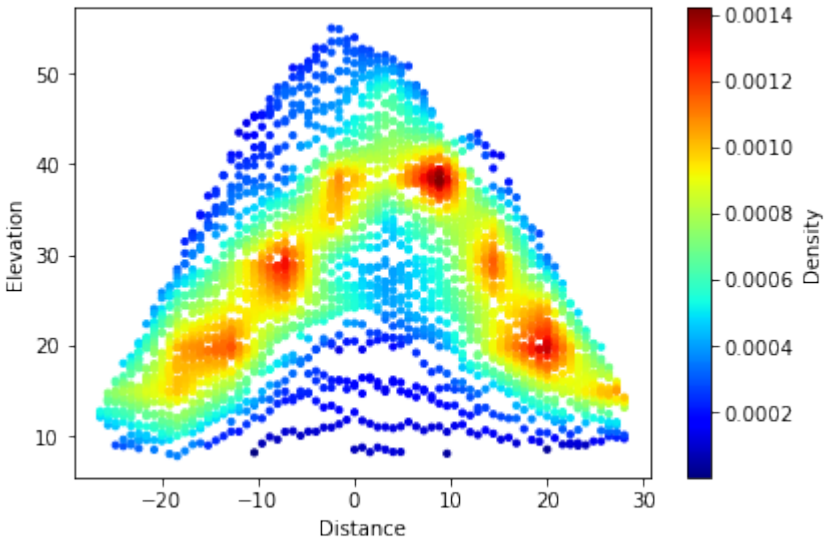
10.4 Step 4. Reclassification with density scatter plot (optional)

[6]:
```
dat = elev.post_tpi(radius=50, min_val=0, swath_plot=False, cross=False, density_
↪scatter=True)
```



10.5 Step 5. Reclassification with cross-swath and density-scatter (optional)

[7]:
```
dat = elev.post_tpi(radius=50, min_val=0, swath_plot=False, cross=True, density_
↪scatter=True, cmap="jet", s=10)
```



CHAPTER

ELEVEN

TOPOGRAPHIC ANALYSIS OF TETON RANGE, WYOMING

The objective of this tutorial is to present the process of analysis shown in the PyOSP paper section 4.1. We will reproduce all relevant results by the following steps.

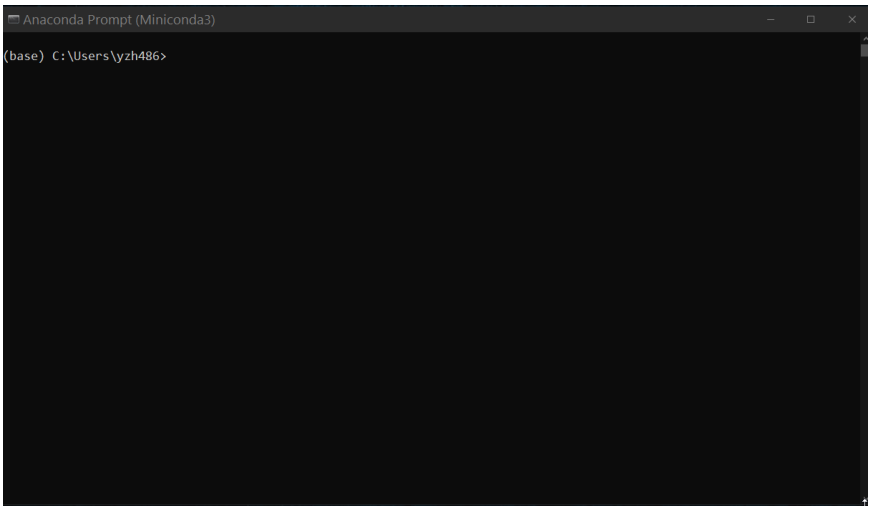
This tutorial starts from installing Python and PyOSP on your computer. However, if you are familiar with the installing process, please go into the section of downloading the dataset.

11.1 Install PyOSP

We start from installing Python on your machine. Because PyOSP is installed through Conda environment, we recommend users install Python through conda environment as well. Firstly, click the link below and download the latest version of Miniconda based on your OS.

https://docs.conda.io/en/latest/miniconda.html

After installing the Miniconde, user should be able to see the Anaconda prompt in the terminal. For example, if Windows is the using OS, searching and clicking the "Anaconda prompt" in the searching bar, it should open a terminal window such as:



Now, we create a conda environment called *env_pyosp*, and install *PyOSP* in it. Entering following commends line by line:

```
conda create -n env_pyosp
conda activate env_pyosp
conda config --env --add channels conda-forge
```

(continues on next page)

(continued from previous page)

```
conda config --env --set channel_priority strict
conda install python=3 pyosp
```

After PyOSP installed, we can verify the installation by entering a *Python* shell (type python in the terminal window), and then check:

[1]:
```
import pyosp
print(pyosp.__version__)
```

```
0.1.6
```

Quit the Python shell by pressing Ctrl-Z and then Enter.

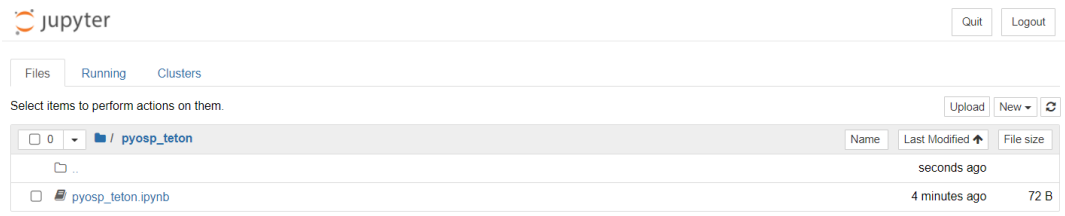
We also recommend install *Jupyter Notebook* for the analysis, run the commend below in the terminal:

```
conda install jupyterlab
```

After you have installed the Jupyter Notebook on your computer, you can start the notebook server by running:

```
jupyter notebook
```

When the notebook opens in your browser, you will see the highest level directory containing notebooks. Now we can create a new folder (from the *new* button on the top right of page) for our analysis, rename it as 'pyosp_teton', and then enter the new folder. In the new folder, you can click the *new* button again, select *Python 3*, it will open a new notebook, such as below:



By entering the notebook, we can start our analysis.

11.2 Download the datasets

First, you will need to download the GeoRaster and baseline shapefile for the swath analysis. In the terminal, *cd* to the folder 'pyosp_teton' that we created above, and input:

```
git clone https://github.com/PyOSP-devs/pyosp-case-studies.git
```

It will download datasets for this analysis.

11.3 Basic swath analysis

In the jupyter notebook, input follwing code snippet:

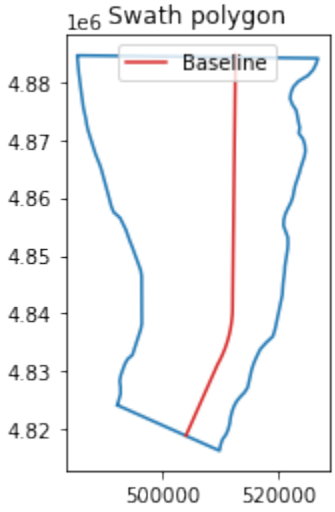
[3]:
```
import pyosp

baseline = './pyosp-case-studies/teton/baseline.shp'
raster = './pyosp-case-studies/teton/teton_dem.tif'

orig = pyosp.Orig_curv(
    baseline,
    raster,
    width=100000,
    line_stepsize=1000,
    cross_stepsize=10
)
```

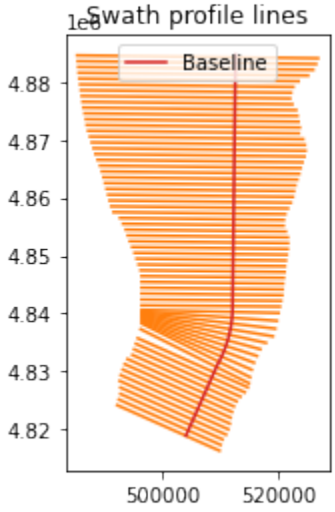
```
Processing: [#########################] 68 of 68 lineSteps
```

After the processing bar went through the end, a swath object *orig* should be generated. Before we move to the swath results, let's take a look at baseline swath polygon, and swath polyline objects.

[4]:
```
import matplotlib.pyplot as plt
from pyosp import read_shape

# read the baseline shape
line_shape = read_shape(baseline)
lx, ly = line_shape.xy

# Plot the swath polygon
fig, ax = plt.subplots()
swath_polygon = orig.out_polygon()
px, py = swath_polygon.exterior.xy
ax.plot(px, py)
ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath polygon")
ax.legend()

# Plot the swath profile lines
fig, ax = plt.subplots()
swath_polylines = orig.out_polylines()
for line in swath_polylines:
    x, y = line.xy
    ax.plot(x, y, color='C1')

ax.plot(lx, ly, color='C3', label="Baseline")
ax.set_aspect('equal', adjustable='box')
ax.set_title("Swath profile lines")
ax.legend()
```

[4]:
```
<matplotlib.legend.Legend at 0x2dd620c97c0>
```

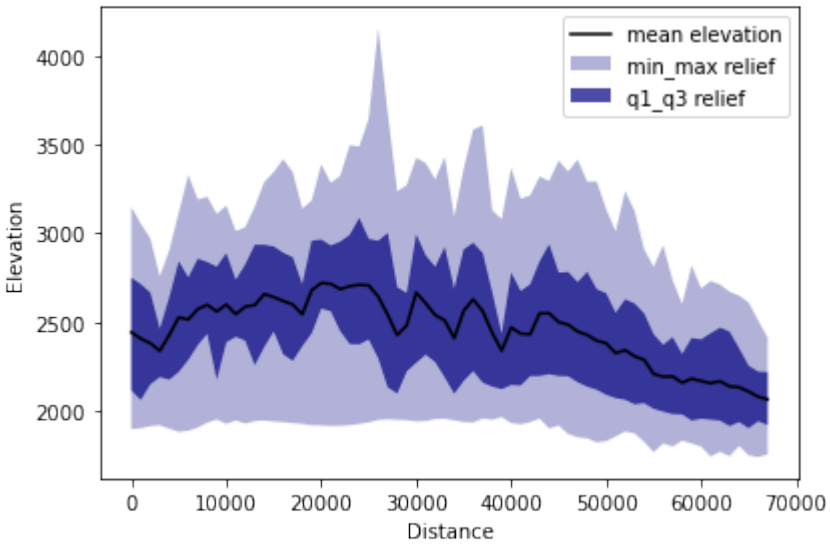




As shown above, our sampling profile lines are bounded by the swath polygon rather than defined swath width 100,000. Because in this case, we used a customized shape associating with the structure of Teton Range to clip the DEM. And our objective is to characterize Teton Range within the structure area. Thus, we used an over-extended swath width 100,000 to ensure the profile line can reach the raster boundary. When PyOSP detected the data boundary of DEM, it will prevent the profile line extending to the nodata region. The resulted sampling data is exactly within the area of clip shape. Check the background information in the PyOSP paper PyOSP paper.

To plot the swath profile, simply run the method:

[30]:
```
orig.profile_plot()
```

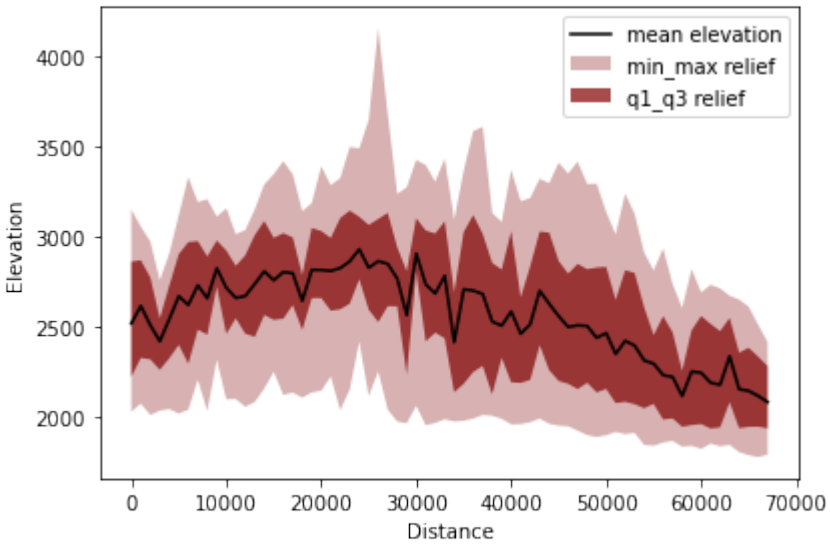


11.4 Differential swath analysis

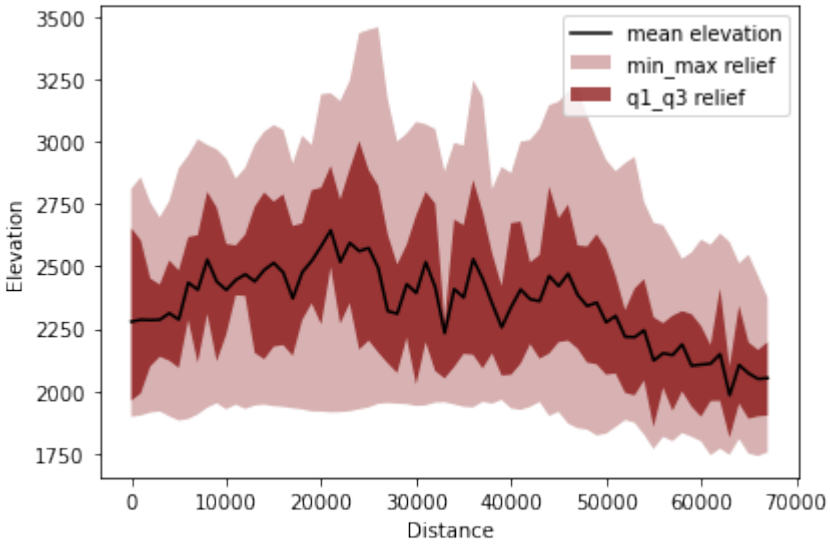
In the paper, we show the ability of PyOSP to split swath results, which we term "differential swaths." We used TPI = 0 during post-processing as a threshold to delineate the inflection of slope shape where topography changes from concave-down (TPI > 0) to convex up (TPI < 0). Thus, differential swaths enable the separation of topography via geomorphic processes, with positive and negative TPI representing erosional and depositional areas of the topography, respectively.

We plot reclassfied TPI>0 and TPI<0 swath profiles as below.

[32]:
```
tpi_big = orig.post_tpi(radius=1000, min_val=0, color='maroon', swath_plot=True)
```



[33]:
```
tpi_small = orig.post_tpi(radius=1000, max_val=0, color='maroon', swath_plot=True)
```



Variables `tpi_big` and `tpi_small` contain the distance and values of reclassified data.

check details in: https://pyosp.readthedocs.io/en/latest/pyosp.curvsp.html#pyosp.curvsp.Base_curv.post_tpi

11.5 Cross swath profile

Next, we evaluate five cross swath profiles along the striking of the baseline. There are two ways to define the boundary of cross swath profile:

1. Specify the start and end points positions from the starting point of baseline.
2. Draw the points along the baseline, and then pass the shapefile of points to the function.

In the documentation of PyOSP, there is a tutorial about application process of these two methods.

https://pyosp.readthedocs.io/en/latest/notebooks/cross_swath.html

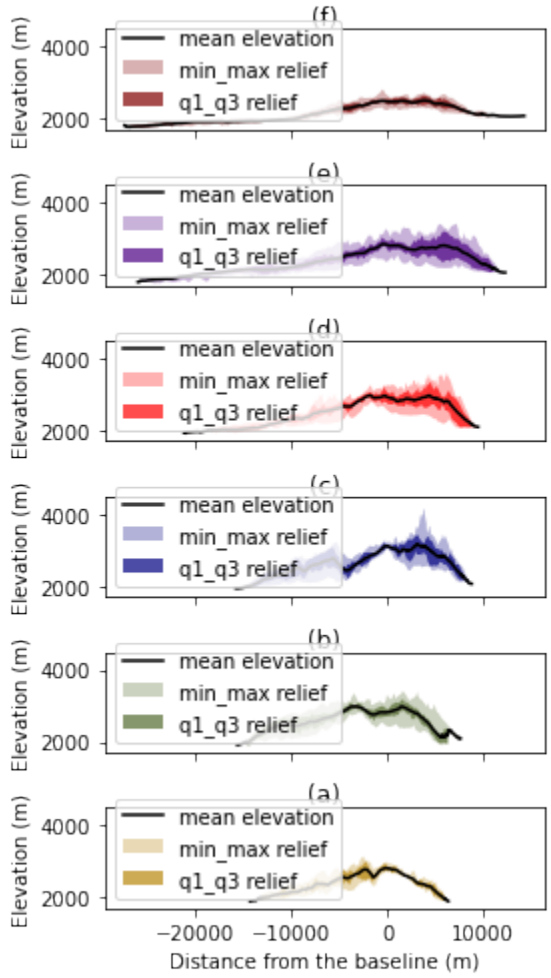
Here, we use distance to define the cross boundary.

[7]:
```
cases = [[58250, None], [42500, 58250], [30000, 42500],
         [22000, 30000], [5000, 22000], [0, 5000]]
colors = ['maroon', 'indigo', 'red', 'navy', 'darkolivegreen',
          'darkgoldenrod']
titles = ['(f)', '(e)', '(d)', '(c)',  '(b)', '(a)']

fig, ax = plt.subplots(6, 1, sharex=True, figsize=(4, 7))
fig.subplots_adjust(hspace=0.2)

for i in cases:
    ind = cases.index(i)
    orig.cross_plot(start=i[0], end=i[1], ax=ax[ind], color=colors[ind])

    ax[ind].set_title(titles[ind], pad=2)
    ax[ind].set_xlabel("")
    ax[ind].set_ylabel("Elevation (m)")
```

(continues on next page)

(continued from previous page)

```
    ax[ind].set_xticks([-20000, -10000, 0, 10000])
    ax[ind].set_ylim(1700,4500)

    if ind == len(cases)-1:
        ax[ind].set_xlabel('Distance from the baseline (m)')

plt.tight_layout()
```



11.6 Summary

This tutorial introduces the swath analysis of Teton range by PyOSP, which corresponds to section 4.1 of PyOSP paper. In the next tutorial, we will use PyOSP to correlate river terraces exemplified by Licking River, Kentucky.

If you have any questions, you can open an issue at https://github.com/PyOSP-devs/PyOSP, or send email to yichuan211@gmail.com

CHAPTER

TWELVE

TERRACE CORRELATION ALONG THE LICKING RIVER, KENTUCKY

In this tutorial, we will show how to use PyOSP performing intelligent swath analysis towards fluvial system. It consists of TPI bases swath analysis, scatter plot of obtained data, and histogram analysis. The content introduced here corresponds to the section 4.2 of PyOSP paper. The installation of PyOSP is introduced by the tutorial *Topographic analysis of Teton Range, Wyoming*, which will not be repeated here.

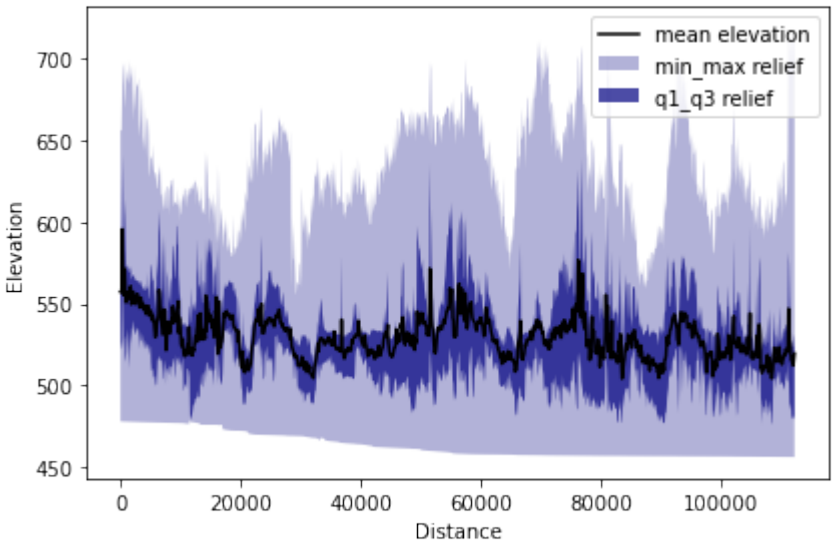
12.1 TPI based swath analysis

In this case study, we use TPI based swath method to characterize the Licking River. In the PyOSP paper Figure 9, we present the differences between TPI based and elevation based methods for identifying the river valley. In general, PyOSP classification of the Licking River Valley was successful using both TPI and elevation (Fig. 9c). However, elevation would not be an ideal classification criterion in valley settings with higher gradients, including a longer reach of the Licking River. A TPI value of 0, on the other hand, will remain the approximate midpoint (inflection point) of the valley slopes along a river's entire length.

[3]:
```
import pyosp

baseline = './pyosp-case-studies/licking/baseline.shp'
raster = './pyosp-case-studies/licking/licking_dem.tif'

tpi = pyosp.Tpi_curv(
    baseline,
    raster,
    width=10000,
    tpi_radius=5000,
    max_tpi=0,
    line_stepsize=100,
    cross_stepsize=100
)
```

```
Processing: [########################] 1123 of 1123 lineSteps
```

[4]:
```
tpi.profile_plot()
```



12.2 Terrace correlation

In this study, PyOSP post-processing capabilities were used to highlight the TPI based swath dataset in a way that accentuated terrace-tread signatures. To this end, the TPI swath area was categorized and filtered to incorporate only those elevation datapoints that also have a slope less than 5 degree, which were then used for a series of scatter-swath, cross-swath, and histogram analyses.

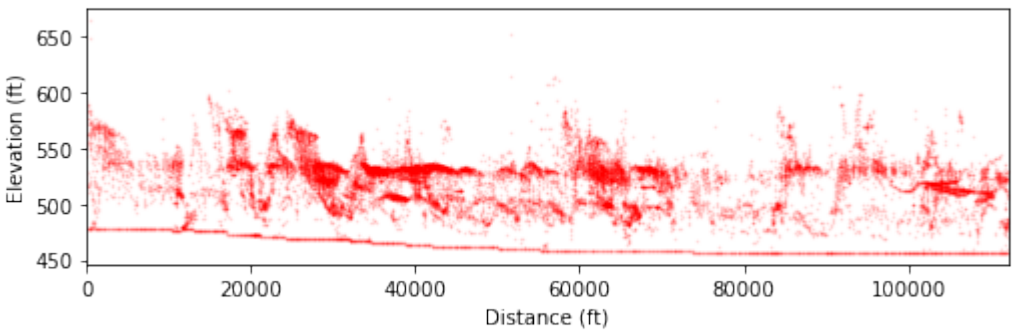
12.2.1 Scatter-swath

[7]:
```
import matplotlib.pyplot as plt
import numpy as np

# swath data with slope less than 5 degree
# see also: https://pyosp.readthedocs.io/en/latest/pyosp.curvsp.html#pyosp.curvsp.Base_
↪curv.post_slope
tpi_small = tpi.post_slope(max_val=5)

distance = tpi_small[0]
dat = tpi_small[1]

fig, ax = plt.subplots(figsize=(7.25, 2.5))

for ind, d in enumerate(distance):
    y = dat[ind]
    x = np.repeat(d, len(y))
    ax.scatter(x, y, s=0.5, color='r', alpha=0.1)

ax.set_xlabel("Distance (ft)")
ax.set_ylabel("Elevation (ft)")
ax.set_xlim([min(distance), max(distance)])
```

(continues on next page)

(continued from previous page)

```
# ax.set_ylim([450, 750])
plt.tight_layout()
```



In the figure above, the dark red clusters correspond to terrace tread elevations as we identified in the field. Details are presented in Figure 10 of PyOSP paper.

12.2.2 Cross-swath profiles

Four cross-swath sections were selected to examine the topographic view of each area.

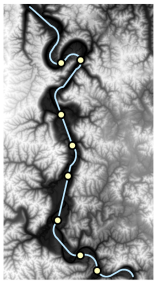
As we mentioned in the Teton case study, there are two ways to define the boundary of cross swath profile:

1. Specify the start and end points positions away from the starting point of baseline.
2. Draw the points along the baseline, and then pass the shapefile of points to the function.

In the documentation of PyOSP, there is a tutorial about application process of these two methods.

https://pyosp.readthedocs.io/en/latest/notebooks/cross_swath.html

Here we choose the second method to define the cross-swath boundary, using the boundary points we drew on the baseline.



[15]:
```
from pyosp.util import point_coords, grouped

# Read boundary coordinates
bound = './pyosp-case-studies/licking/BoundaryPoints.shp'
bound_coords = point_coords(bound)

# cross-post-processing
colors = ['maroon', 'darkgreen', 'indigo', 'red']
titles = ['cross_1', 'cross_2',  'cross_3', 'cross_4']
ind = 0
```

(continues on next page)

(continued from previous page)

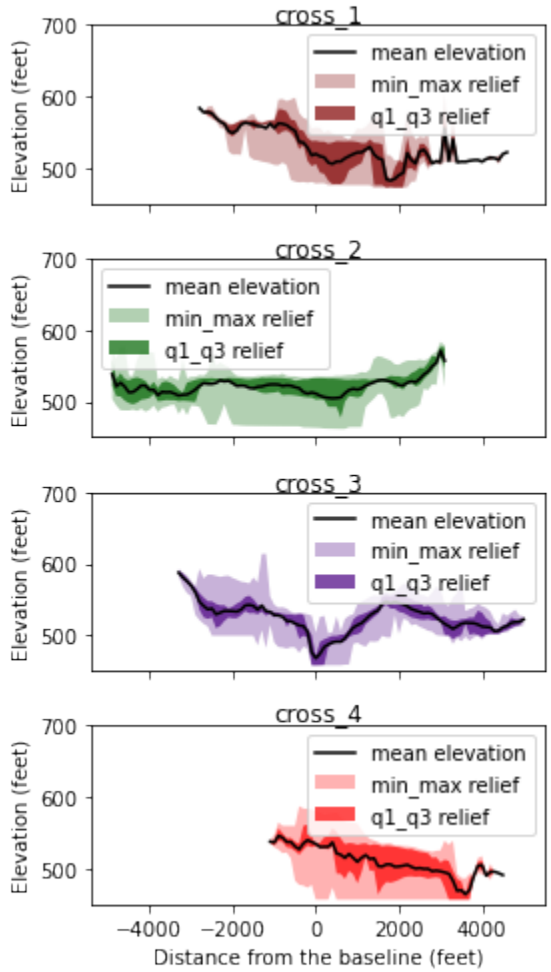
```
fig, ax = plt.subplots(4, 1, sharex=True, figsize=(4, 7))
fig.subplots_adjust(hspace=0.2)

for start, end in grouped(bound_coords):
    tpi.post_slope(
        max_val=5,
        start=start,
        end=end,
        ax=ax[ind],
        color=colors[ind],
        cross=True,
        swath_plot=True
    )
    ax[ind].set_title(titles[ind], pad=2)
    ax[ind].set_xlabel("")
    ax[ind].set_ylabel("Elevation (feet)")
    ax[ind].set_ylim(450,700)

    if ind == len(colors)-1:
        ax[ind].set_xlabel('Distance from the baseline (feet)')

    ind += 1

plt.tight_layout()
```

```
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪171: RuntimeWarning: All-NaN axis encountered
  min_z = [np.nanmin(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪172: RuntimeWarning: All-NaN axis encountered
  max_z = [np.nanmax(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪173: RuntimeWarning: Mean of empty slice
  mean_z = [np.nanmean(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\numpy\lib\nanfunctions.py:
↪1368: RuntimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪171: RuntimeWarning: All-NaN axis encountered
  min_z = [np.nanmin(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪172: RuntimeWarning: All-NaN axis encountered
  max_z = [np.nanmax(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪173: RuntimeWarning: Mean of empty slice
  mean_z = [np.nanmean(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\numpy\lib\nanfunctions.py:
↪1368: RuntimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪171: RuntimeWarning: All-NaN axis encountered
```

(continues on next page)

(continued from previous page)

```
  min_z = [np.nanmin(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪172: RuntimeWarning: All-NaN axis encountered
  max_z = [np.nanmax(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪173: RuntimeWarning: Mean of empty slice
  mean_z = [np.nanmean(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\numpy\lib\nanfunctions.py:
↪1368: RuntimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪171: RuntimeWarning: All-NaN axis encountered
  min_z = [np.nanmin(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪172: RuntimeWarning: All-NaN axis encountered
  max_z = [np.nanmax(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\pyosp\curvsp\base_curv.py:
↪173: RuntimeWarning: Mean of empty slice
  mean_z = [np.nanmean(x) if len(x) > 0 else np.nan for x in z]
C:\Users\yzh486\Miniconda3\envs\env_pyosp\lib\site-packages\numpy\lib\nanfunctions.py:
↪1368: RuntimeWarning: All-NaN slice encountered
  r, k = function_base._ureduce(
```



12.2.3 Histogram of cross-swath profile

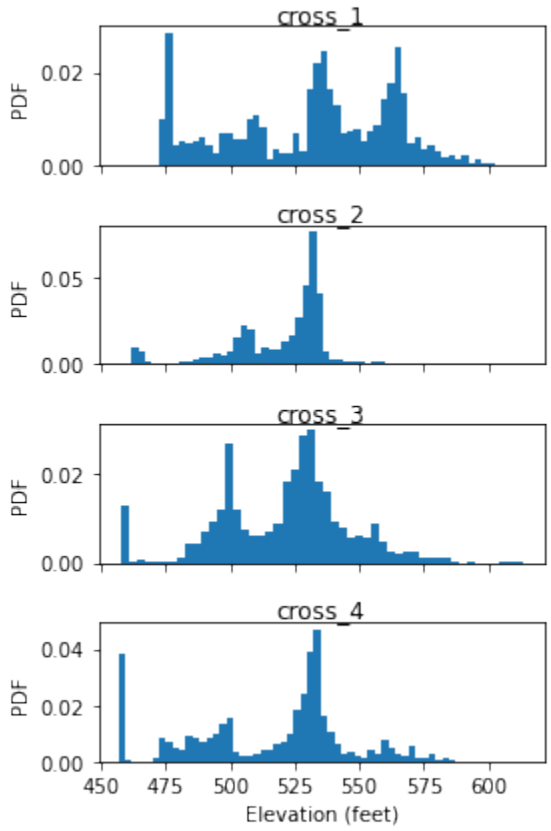
Last, we plot the histogram of captured cross-swath profile.

[16]:
```
ind = 0

fig, ax = plt.subplots(4, 1, sharex=True, figsize=(4, 6))
fig.subplots_adjust(hspace=0.2)

for start, end in grouped(bound_coords):
    dat = tpi.post_slope(max_val=5.,
                         start=start,
                         end=end,
                         ax=ax[ind],
                         color=colors[ind],
                         cross=True,
                         swath_plot=False
                        )
    tpi.hist(dat=dat[1],ax=ax[ind])
```

(continues on next page)

(continued from previous page)

```
    ax[ind].set_title(titles[ind], pad=2)
    ax[ind].set_xlabel("")
    ax[ind].set_ylabel("PDF")

    if ind == len(colors)-1:
        ax[ind].set_xlabel('Elevation (feet)')

    ind += 1

plt.tight_layout()
```



Discussion of above results are presented in the PyOSP paper.

12.3 Summary

This tutorial introduces the swath analysis of Licking River by PyOSP, which corresponds to section 4.2 of PyOSP paper. In the next tutorial, we will use PyOSP to perform circular swath analysis toward Olympus Mons, Mars.

If you have any further questions, you can open an issue at https://github.com/PyOSP-devs/PyOSP, or send email to yichuan211@gmail.com

CHAPTER

THIRTEEN

CIRCULAR SWATH ANALYSIS OF OLYMPUS MONS, MARS

In this tutorial, we will perform circular swath analysis toward the largest known shield volcano in the solar system, Olympus Mons, Mars. It consists of TPI bases swath analysis, slice of the profile, and comparison between traditional and TPI based circular swath profile. The content introduced here corresponds to the section 4.3 of PyOSP paper. The installation of PyOSP is introduced by the tutorial *Topographic analysis of Teton Range, Wyoming*, which will not be repeated here.

13.1 Traditional and TPI-based circular swath profiles

To assess the circular swath capabilities of PyOSP, we performed swath analysis of the Olympus Mons edifice as well as used several post-processing functions to investigate the asymmetry of the volcano. We conducted traditional fixed-radius- and TPI-based swath analysis to compare and assess the efficacy of both methods.

Note that the circular swath profile provided by PyOSP can be also applied to other geographic objects, such as impact craters, sinkholes, seafloor pockmarks, etc.

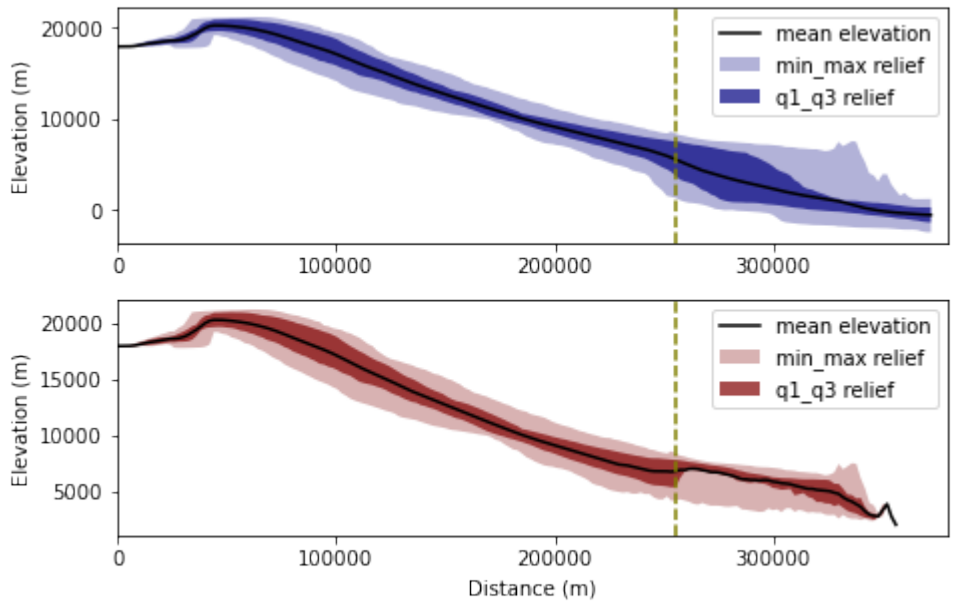
[9]:
```
import pyosp

center = './pyosp-case-studies/olympus/center.shp'
raster = './pyosp-case-studies/olympus/olympus_dem.tif'

orig = pyosp.Orig_cir(
    center,
    raster,
    radius=370000,
    ng_stepsize=1,
    radial_stepsize=2000
)

tpi = pyosp.Tpi_cir(
    center,
    raster,
    radius=400000,
    ng_start=0,
    ng_end=360,
    tpi_radius=300000,
    min_tpi=0,
    ng_stepsize=1,
    radial_stepsize=2000
)
```

```
Processing: [########################] 360 of 360 lineSteps
```

13.2 Compare swath profiles

Now we can compare swath profiles based on TPI and traditional methods

[10]:
```
import matplotlib.pyplot as plt

cases = [orig, tpi]
colors = ['navy', 'maroon']

fig, ax = plt.subplots(2, 1, figsize=(6.8,4.4))
fig.subplots_adjust(hspace=0.2)

for i in range(len(cases)):
    cases[i].profile_plot(ax=ax[i], color=colors[i])
    ax[i].axvline(x=255000, color='olive', lw=1.5, linestyle='--')
    ax[i].set_xlabel("")
    ax[i].set_ylabel("Elevation (m)")
    ax[i].set_xticks([0, 100000, 200000, 300000])
    ax[i].set_xlim([0,380000])

    if i == len(cases)-1:
        ax[i].set_xlabel("Distance (m)")
```



The calibrated polygons overlapping the hillshade of Olympus Mons are shown in Figure 12 of PyOSP paper. Results show that the TPI-based method accurately delineates the edifice along the boundary of the basal scarp, whereas the fixed-radius swath invariably generated a circular area that includes unwanted areas beyond the volcano.The pronounced fall in IQR at 255,000m on the TPI-based swath signals a reduction in datapoints along the southeastern basal scarp and marks the boundary of asymmetry.

13.2.1 Slice plot across the valcano

Next, we used PyOSP to plot four topographic profiles across the volcano.

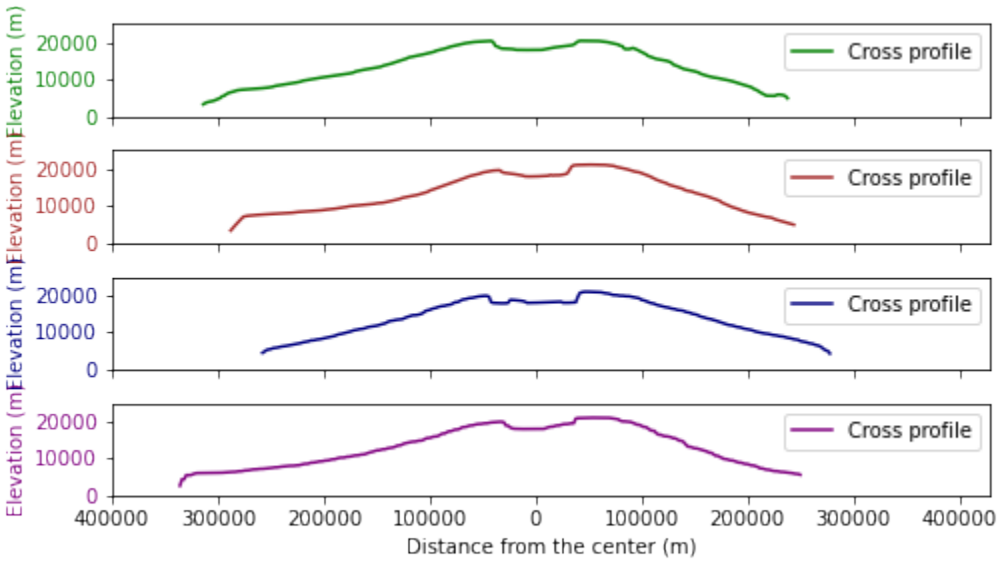
[12]:
```
import numpy as np

angles = [0, 180, 270, 90, 225, 45, 315, 135]

sector = np.arange(tpi.ng_start, tpi.ng_end+1e-10,
                   tpi.ng_stepsize)

value_all = []
d_all = []

for i in angles:
    ng_ind = np.abs(sector - i).argmin()
    points = tpi.lines[ng_ind]
    values = tpi.dat[ng_ind]
    values = values[~np.isnan(values)]

    d = []
    for point in points:
        d_temp = np.sqrt((point[0]-points[0][0])**2
                         +(point[1]-points[0][1])**2)
        d.append(d_temp)

    value_all.append(values)
    d_all.append(d)

fig, ax = plt.subplots(4, 1, figsize=(7.2,4))
fig.subplots_adjust(hspace=0)

ax_num = 0
colors = ['green', 'brown', 'navy', 'purple']

for x, y in zip(*[iter(range(len(angles)))] * 2):
    d_r = d_all[x]
    d_l = [-1 * i for i in d_all[y]][::-1]
    value_r = value_all[x]
    value_l = value_all[y][::-1]

    d = d_l + d_r[1:None]
    value = np.concatenate((value_l, value_r[1:None]))

    ax[ax_num].plot(d, value, color=colors[ax_num], label="Cross profile")
    ax[ax_num].set_xlim([-400000, 430000])
    ax[ax_num].set_ylim([0, 25000])
    ax[ax_num].set_yticks([0, 10000, 20000])
    ax[ax_num].tick_params(axis='y', labelcolor=colors[ax_num])
    ax[ax_num].set_xticklabels([])
    ax[ax_num].set_ylabel("Elevation (m)", color=colors[ax_num])
    ax[ax_num].legend()
```

(continues on next page)

(continued from previous page)

```
    if ax_num == len(colors)-1:
        ticks = ax[ax_num].get_xticks()
        ax[ax_num].set_xticklabels([int(abs(tick)) for tick in ticks])
        ax[ax_num].set_xlabel('Distance from the center (m)')

    ax_num += 1

plt.tight_layout()
```

```
<ipython-input-12-aef1d7cad2a0>:52: UserWarning: FixedFormatter should only be used␣
↪together with FixedLocator
  ax[ax_num].set_xticklabels([int(abs(tick)) for tick in ticks])
```



13.3 Summary

This tutorial introduces the swath analysis of Olympus Mons by PyOSP, which corresponds to section 4.3 of PyOSP paper.

If you have any questions, you can open an issue at https://github.com/PyOSP-devs/PyOSP, or send emails to yichuan211@gmail.com

Cheers!

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search