polylx

Ondrej Lexa

Mar 05, 2024

CONTENTS

Contents:

CHAPTER

ONE

POLYLX - PYTHON PACKAGE TO VISUALIZE AND ANALYZE DIGITIZED 2D MICROSTRUCTURES

1.1 Installation

1.1.1 PyPI

To install PolyLX, just execute

```
pip install polylx
```

Upgrading via pip

To upgrade an existing version of PolyLX from PyPI, execute

```
pip install polylx --upgrade --no-deps
```

Please note that the dependencies (Matplotlib, NumPy, Pandas, NetworkX, seaborn, shapely, pyshp and SciPy) will also be upgraded if you omit the `--no-deps` flag; use the `--no-deps` ("no dependencies") flag if you don't want this.

Installing PolyLX with conda or mamba

Another common way to install is create environment using conda or mamba. Download latest version of polylx and unzip to folder of your choice. Use conda or mamba to create an environment from an `environment.yml` file. Open the terminal, change directory where you unzip the source and execute following command:

```
conda env create -f environment.yml
```

Activate the new environment and install from current directory:

```
conda activate polylx
pip install polylx
```

1.2 Documentation

Explore the full features of PolyLX. You can find detailed documentation here.

1.3 Contributing

Most discussion happens on Github. Feel free to open an issue or comment on any open issue or pull request. Check `CONTRIBUTING.md` for more details.

1.4 License

CHAPTER

TWO

TUTORIAL

The microstructural analysis is a powerful, but underused tool of petrostructural analysis. Except acquirement of common statistical parameters, this technique can significantly improve understanding of processes of grain nucleation and grain growth, can bring insights on the role of surface energies or quantify duration of metamorphic and magmatic cooling events as long as appropriate thermodynamical data for studied mineral exist. This technique also allows systematic evaluation of degree of preferred orientations of grain boundaries in conjunction with their frequencies. This may help to better understand the mobility of grain boundaries and precipitations or removal of different mineral phases.

We introduce a new platform, object-oriented Python package PolyLX providing several core routines for data exchange, visualization and analysis of microstructural data, which can be run on any platform supported by Scientific Python environment.

2.1 Grains objects

To start working with PolyLX we need to import the polylx package. For our convinience, we can import PolyLX into actual namespace:

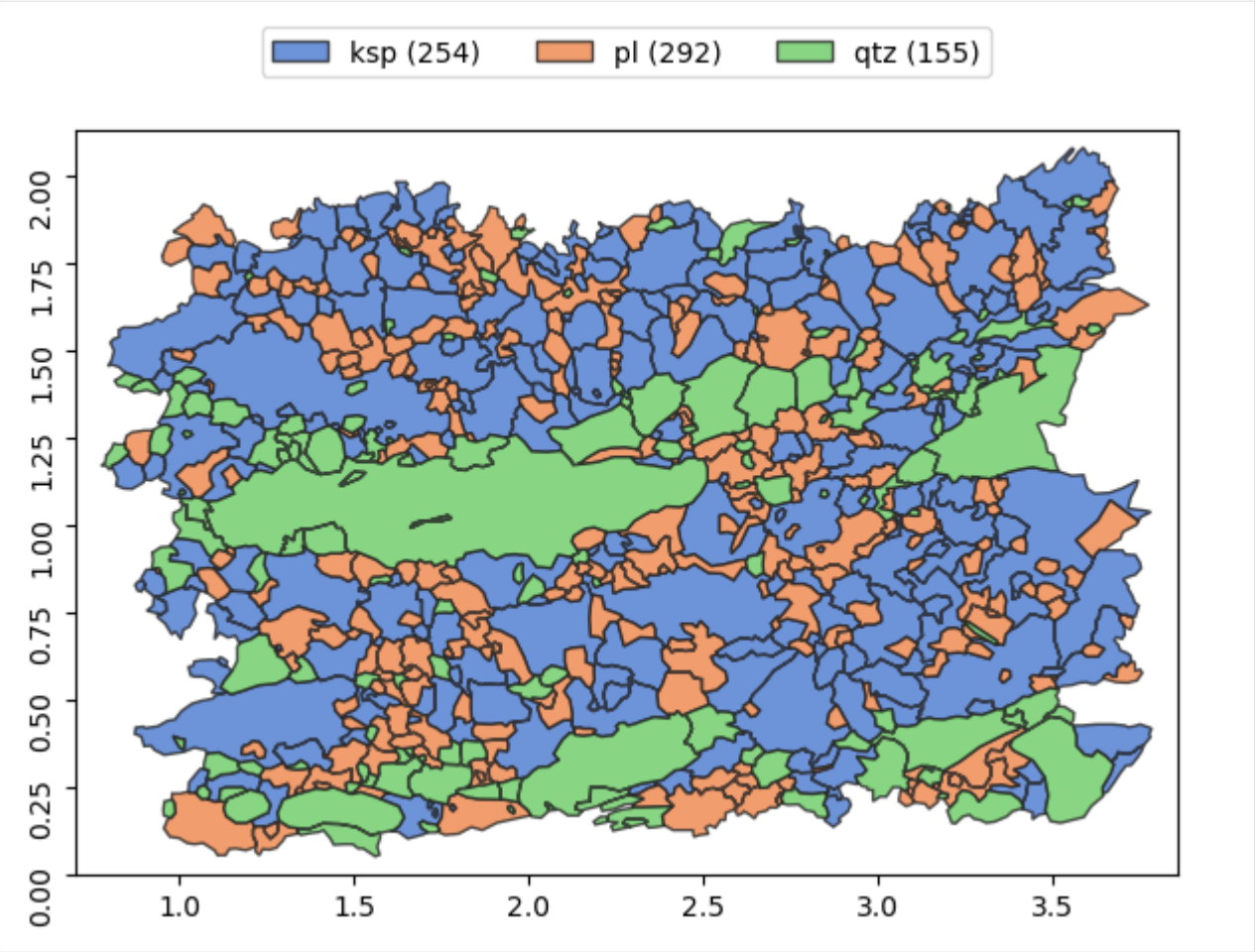
[1]:
```
from polylx import *
```

To read example data, we can use `example` method. Note that we create new `Grains` object, which store all imported `Grain` features.

[2]:
```
g = Grains.example()
```

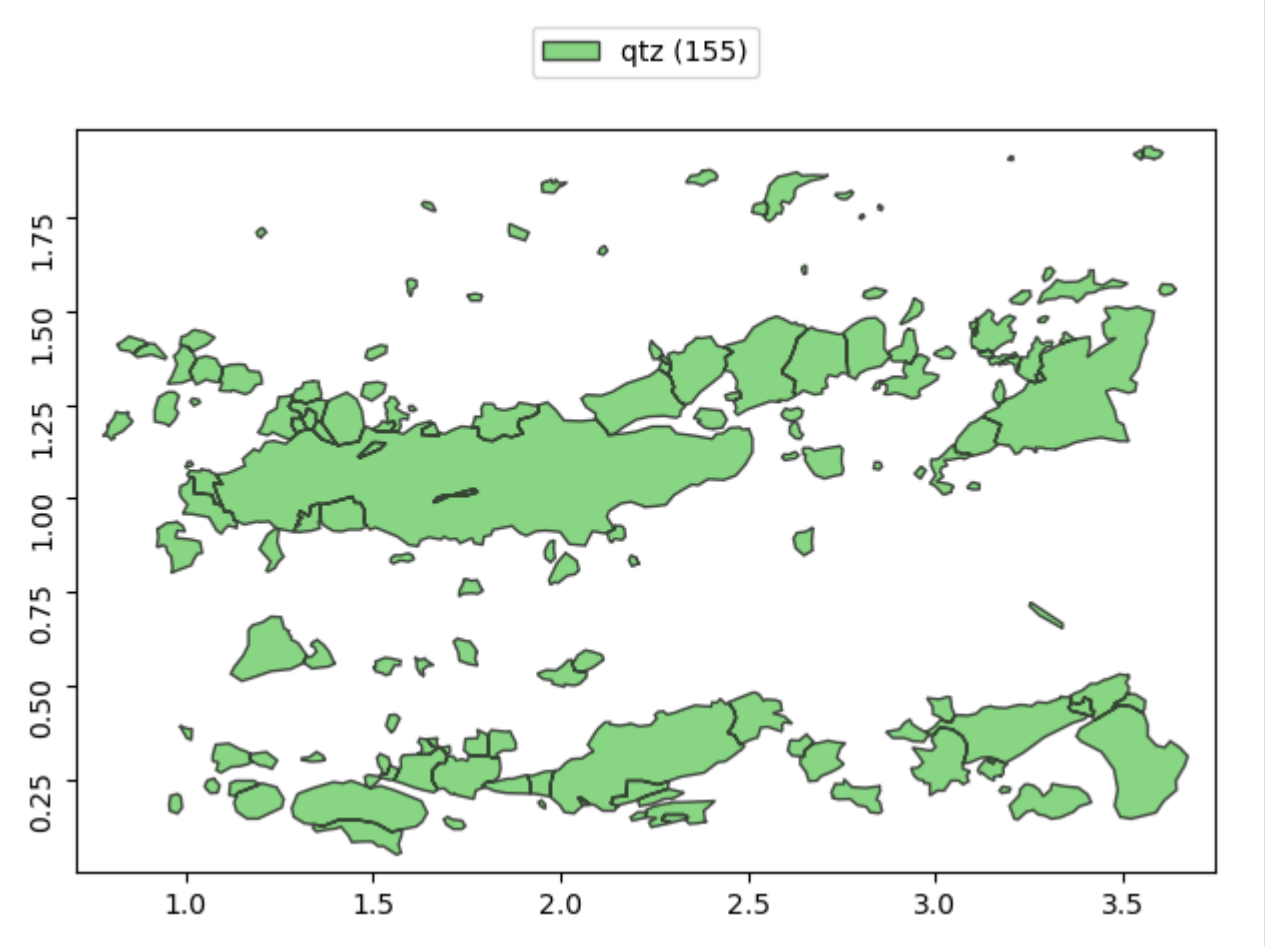
To visualize grain objects from shape file, we can use `plot` method of `Grains` object:

[3]:
```
g.show()
```



or we can just select `Grains` by its class name:

[4]:
```
g['qtz'].show()
```



The dot notation is used to access individual properties commonly returning values for individual `Grain` as `numpy` array.

[5]:
```
g['qtz'].ar
```

[5]:
```
array([1.46370088, 3.55371458, 1.43641139, 1.26293055, 2.10676277,
       1.45200805, 1.98973326, 1.97308557, 2.13420187, 1.76682269,
       1.70083897, 1.38205897, 1.88811465, 1.59948827, 2.50452919,
       1.60296389, 1.4918233 , 2.15318719, 1.27665794, 1.38714959,
       1.67235338, 2.33179583, 1.30609967, 2.73148246, 1.02760669,
       1.33627299, 2.65451284, 1.29069569, 1.73051094, 1.25763409,
       1.90027316, 2.56110638, 1.78555385, 2.40926108, 2.26741705,
       1.71957235, 1.79168709, 1.04770164, 1.293186  , 1.29420065,
       1.48331817, 2.15510614, 2.21246419, 1.57101091, 2.01989715,
       1.1428675 , 2.02888455, 4.07405108, 1.47968881, 1.24770095,
       1.4750185 , 1.37946472, 1.49048108, 1.56668345, 1.43717521,
       1.59756777, 1.58948843, 2.12557437, 2.54316052, 1.98917177,
       1.29809155, 1.70022052, 1.40121941, 1.24674038, 1.50255058,
       1.42880415, 1.73447054, 2.3548111 , 1.52891827, 3.26773221,
       1.33011244, 2.26173396, 3.2151532 , 2.15638456, 1.61602624,
       1.13898611, 2.91625233, 1.94275485, 2.68487563, 1.12446842,
       1.48814907, 1.79425743, 1.19512385, 1.28301942, 1.39853133,
       1.59860483, 3.80709622, 1.75016693, 1.59940152, 1.43972155,
```

(continues on next page)

(continued from previous page)

```
       1.09439109, 2.00023212, 1.87470191, 1.04157011, 1.48561371,
       1.14172901, 1.48211332, 1.52569202, 1.59357336, 1.58054224,
       1.86890813, 1.84729576, 1.45085424, 1.4400654 , 2.6284034 ,
       1.62077026, 1.35218688, 1.69040095, 1.2829313 , 2.7380623 ,
       1.55901231, 1.72569674, 1.18396915, 1.67864861, 2.40971617,
       2.08496427, 2.12907657, 1.20981316, 1.46045276, 1.55428179,
       4.73482949, 2.32570855, 1.95106722, 1.81174297, 4.08295286,
       2.04530043, 1.56215221, 1.42587721, 1.70016792, 1.78887212,
       2.17273986, 2.47995119, 4.59660941, 3.43961286, 3.04193405,
       2.91162332, 2.98790473, 2.55352686, 1.33076709, 7.09385883,
       1.91715238, 1.47161362, 2.39020581, 1.51938795, 1.87839843,
       1.9946499 , 2.27873759, 4.50321651, 5.78162231, 6.9806063 ,
       1.3177092 , 2.33701528, 1.86371784, 1.26166336, 1.28322623])
```

or we can collect any properties to `pandas.DataFrame` using `df` method:

[6]:
```
g.df('la', 'sa', 'lao', 'sao', 'area', 'length', 'ead', 'ar').head(10)
```

[6]:
```
           la        sa         lao         sao      area    length       ead  \
fid
0    0.066027  0.045110   70.596636  160.596636  0.002286  0.186196  0.053956
1    0.099033  0.057029   70.983857  160.983857  0.004409  0.258753  0.074922
2    0.074248  0.020893   61.438248  151.438248  0.001123  0.175821  0.037813
3    0.045232  0.031489   85.088587  175.088587  0.001005  0.134427  0.035779
4    0.136445  0.108038  170.839835   80.839835  0.011489  0.398558  0.120948
5    0.073578  0.044938  123.223347   33.223347  0.002471  0.201258  0.056090
6    0.103567  0.065119  149.397514   59.397514  0.005213  0.283110  0.081474
7    0.103189  0.077988   23.758847  113.758847  0.005951  0.318774  0.087048
8    0.187049  0.036611   82.108720  172.108720  0.004407  0.404066  0.074904
9    0.270513  0.128402   76.193288  166.193288  0.024576  0.729051  0.176894

           ar
fid
0    1.463701
1    1.736522
2    3.553715
3    1.436411
4    1.262931
5    1.637319
6    1.590441
7    1.323142
8    5.109041
9    2.106763
```

[7]:
```
g.df('ead').describe()
```

[7]:
```
              ead
count  701.000000
mean     0.072812
std      0.056812
min      0.000350
25%      0.037140
50%      0.058338
```

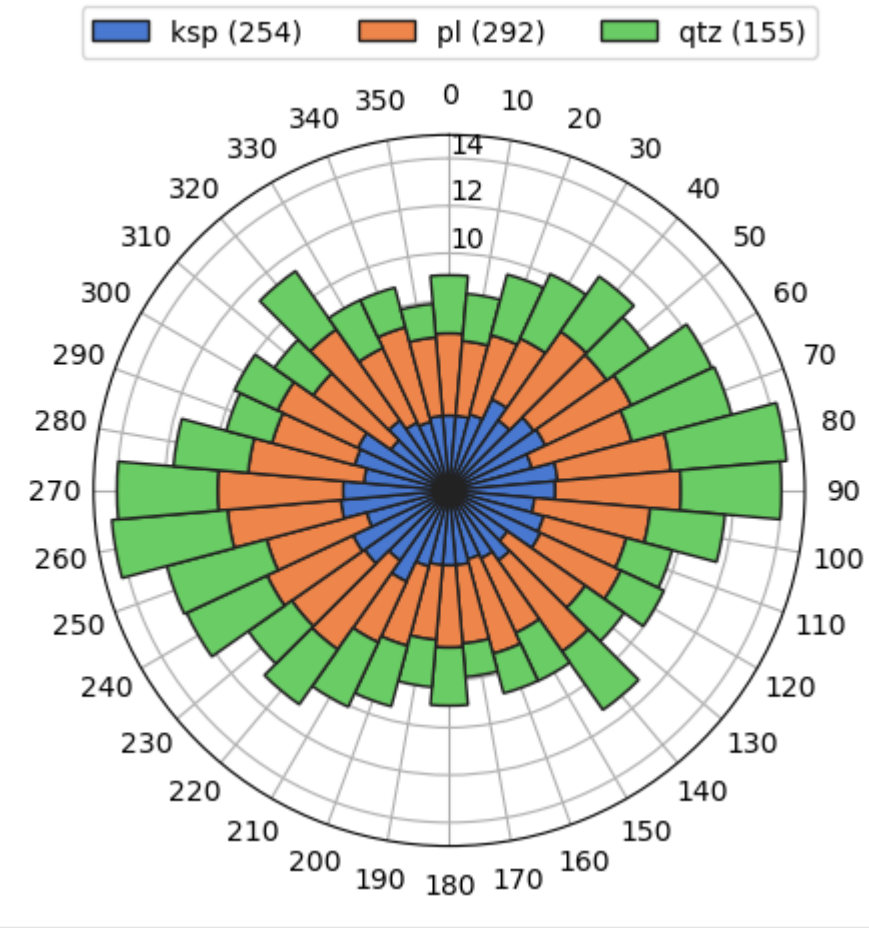
(continues on next page)

(continued from previous page)

```
75%       0.093503
max       0.638144
```

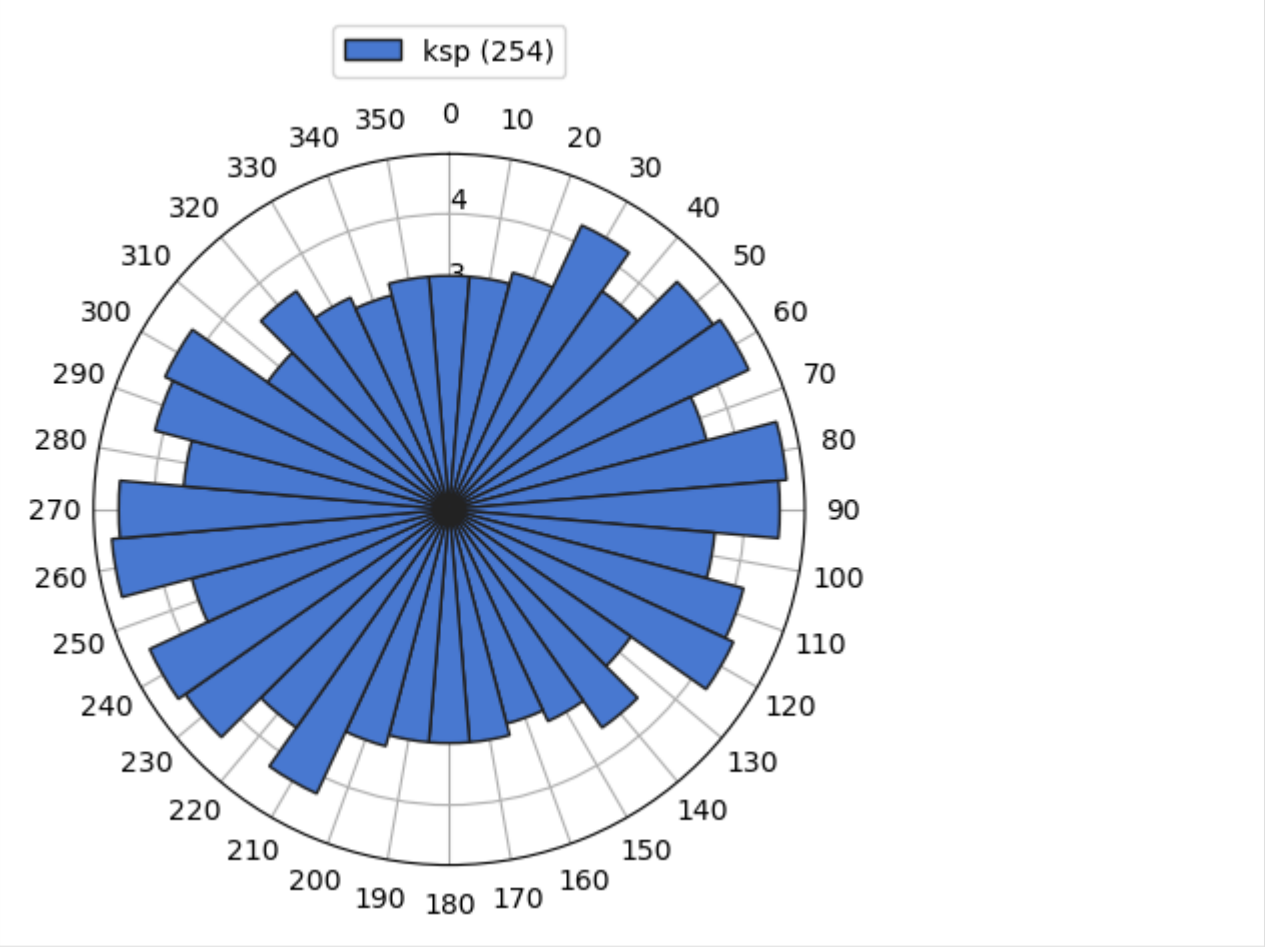
To visualize orientations of objects, we can use `rose` method:

[8]:
```
g.rose()
```



or for just single class:

[9]:
```
g['ksp'].rose()
```



To aggregate and summarize multiple properties by different functions according to defined classification (`name` by default) we can use `agg` method:

[10]:
```
g.agg('name', 'count', 'area', 'sum', 'ead', 'mean', 'lao', circular.mean)
```

[10]:
```
       name      area       ead        lao
class
ksp     254  2.443733  0.089710  76.875488
pl      292  1.083516  0.060629  94.197847
qtz     155  1.166097  0.068071  74.320337
```

The `groups` method return Pandas `GroupBy` object which allows any pandas-style manipulation

[11]:
```
g.groups('ead', 'area', 'la', 'sa').describe().T
```

[11]:
```
class                ksp          pl           qtz
ead  count  2.540000e+02  292.000000  1.550000e+02
     mean   8.970974e-02    0.060629  6.807125e-02
     std    6.495077e-02    0.032438  7.054971e-02
     min    6.641998e-04    0.001850  3.501464e-04
     25%    4.133005e-02    0.038226  2.970151e-02
     50%    7.403298e-02    0.053984  4.794577e-02
     75%    1.191733e-01    0.077308  7.892656e-02
     max    4.105520e-01    0.190210  6.381439e-01
```

(continues on next page)

(continued from previous page)

```
area count  2.540000e+02  292.000000  1.550000e+02
     mean   9.620995e-03    0.003711  7.523208e-03
     std    1.548182e-02    0.004170  2.778736e-02
     min    3.464873e-07    0.000003  9.629176e-08
     25%    1.341681e-03    0.001148  6.930225e-04
     50%    4.304819e-03    0.002289  1.805471e-03
     75%    1.115444e-02    0.004694  4.892680e-03
     max    1.323812e-01    0.028416  3.198359e-01
la   count  2.540000e+02  292.000000  1.550000e+02
     mean   1.295772e-01    0.086681  1.019395e-01
     std    1.053259e-01    0.053220  1.366152e-01
     min    1.013949e-03    0.006461  1.017291e-03
     25%    5.439610e-02    0.050202  4.314167e-02
     50%    9.871911e-02    0.072777  7.151284e-02
     75%    1.793952e-01    0.106761  1.206513e-01
     max    8.097226e-01    0.279398  1.437277e+00
sa   count  2.540000e+02  292.000000  1.550000e+02
     mean   7.545538e-02    0.049585  5.255111e-02
     std    5.428555e-02    0.027663  4.632415e-02
     min    3.648908e-04    0.000583  1.457310e-04
     25%    3.370683e-02    0.031980  2.183093e-02
     50%    6.643814e-02    0.043545  3.640605e-02
     75%    1.021515e-01    0.063468  6.490102e-02
     max    3.252086e-01    0.166726  3.035541e-01
```

The method `classify` could be used to define new classification, based on any property and using variety of rules, e.g. `'quantile'`:

[12]:
```
g.classify('ar', rule='quantile', k=6)
df = g.df('class', 'name', 'area')
df.head()
```

[12]:
```
          class name      area
fid
0     1.45-1.63  qtz  0.002286
1     1.63-1.89   pl  0.004409
2    2.36-12.16  qtz  0.001123
3     1.28-1.45  qtz  0.001005
4     1.02-1.28  qtz  0.011489
```

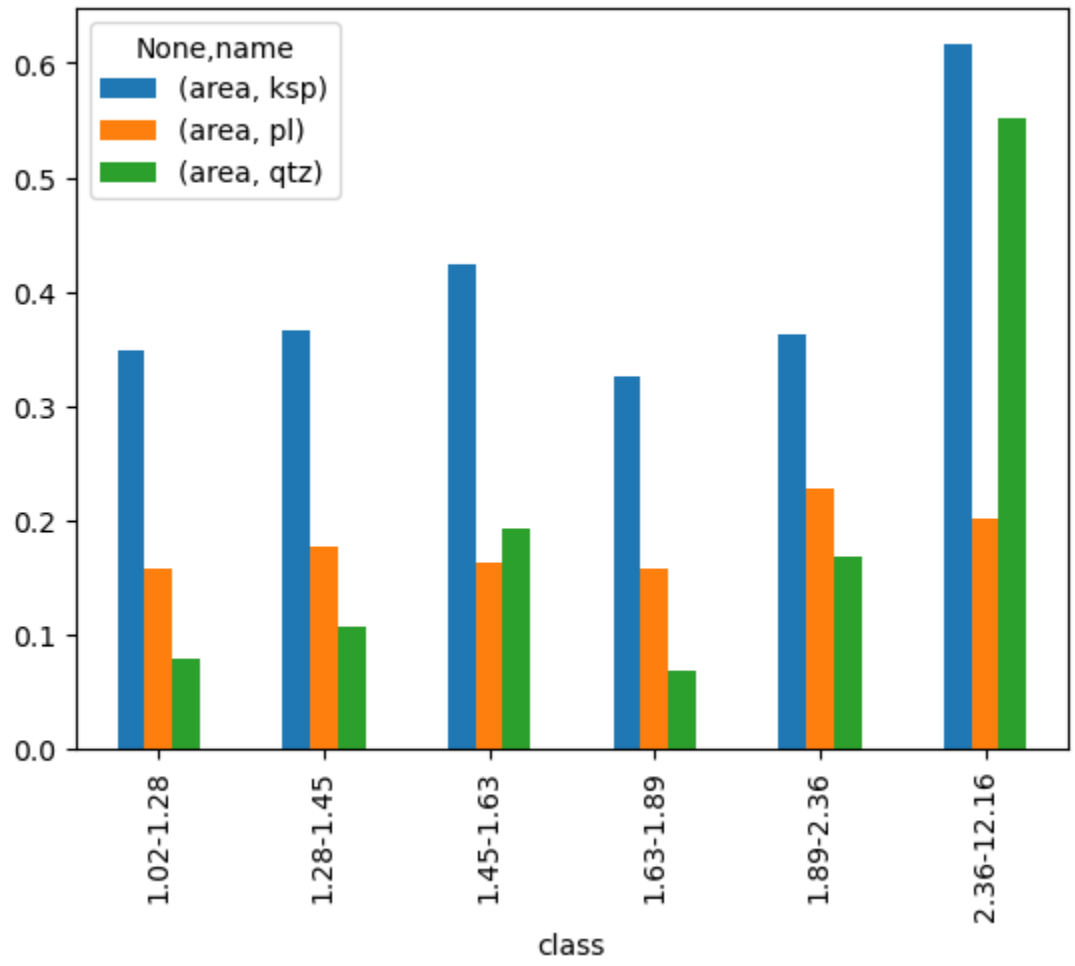
To summarize results for individual phases per class we can use pandas pivot table:

[13]:
```
pd.pivot_table(df,index=['class'], columns=['name'], aggfunc='sum')
```

[13]:
```
               area
name            ksp        pl       qtz
class
1.02-1.28  0.348379  0.158328  0.079026
1.28-1.45  0.366220  0.176028  0.106734
1.45-1.63  0.424773  0.163026  0.192027
1.63-1.89  0.325575  0.157087  0.067631
1.89-2.36  0.362103  0.226971  0.168503
2.36-12.16 0.616684  0.202075  0.552178
```

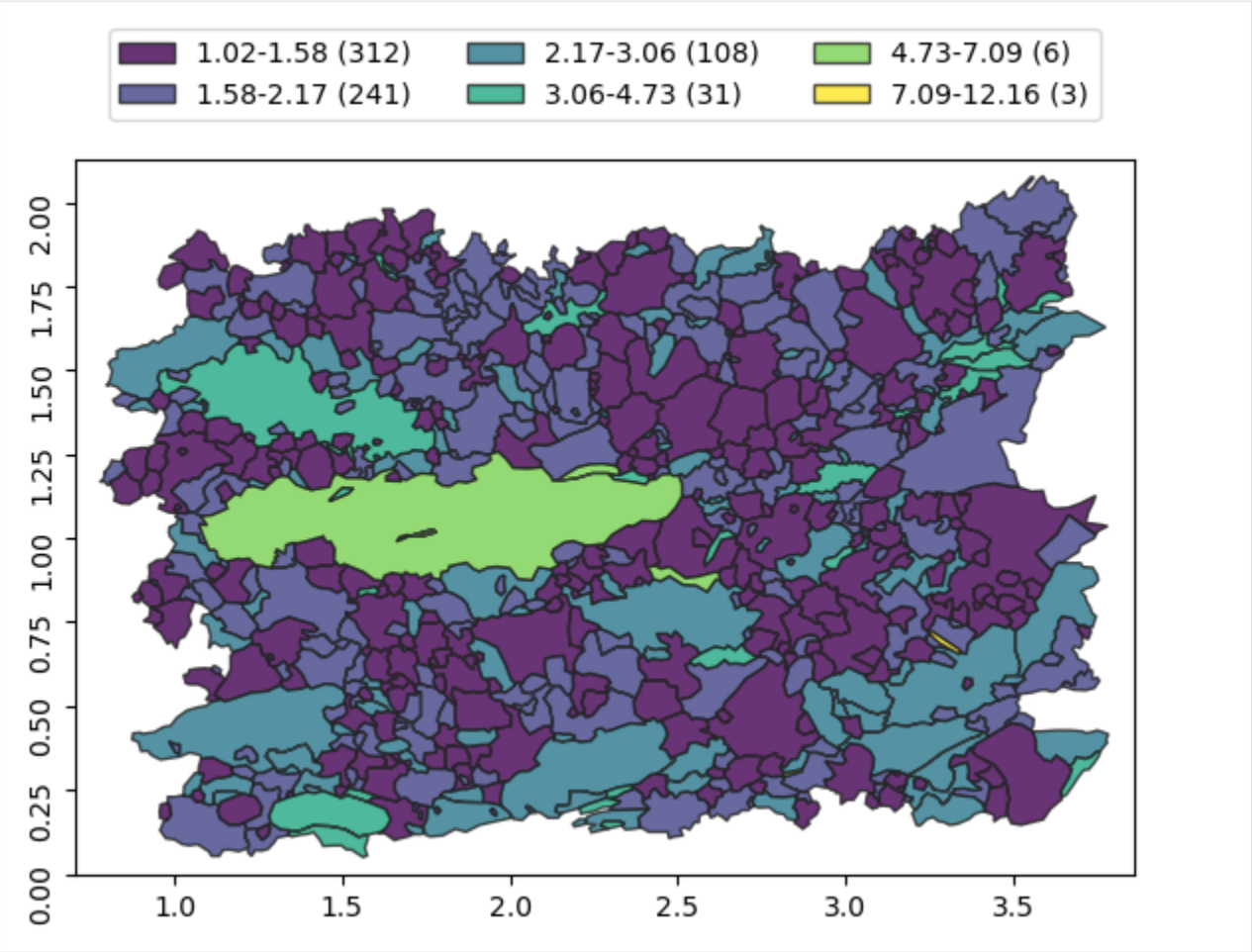
or we can directly plot it..

[14]:
```
pd.pivot_table(df,index=['class'], columns=['name'], aggfunc='sum').plot(kind='bar');
```

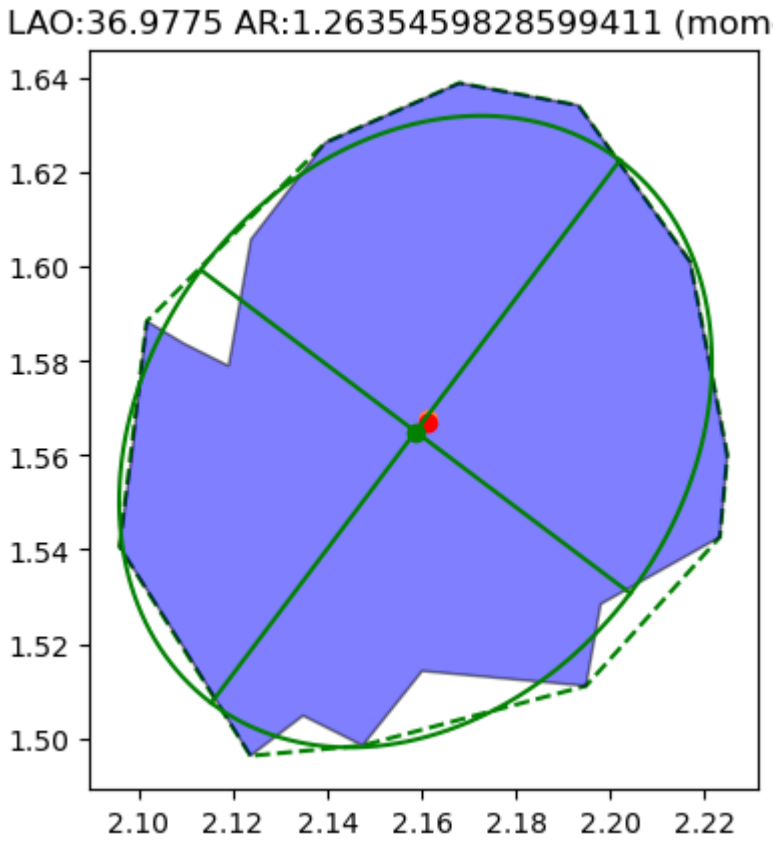


[15]:
```
g.classify('ar', rule='jenks', k=6)
```

[16]:
```
g.show()
```


1.02-1.58 (312)
1.58-2.17 (241)
2.17-3.06 (108)
3.06-4.73 (31)
4.73-7.09 (6)
7.09-12.16 (3)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1.0
1.5
2.0
2.5
3.0
3.5

[17]:
```
g[132].show()
```

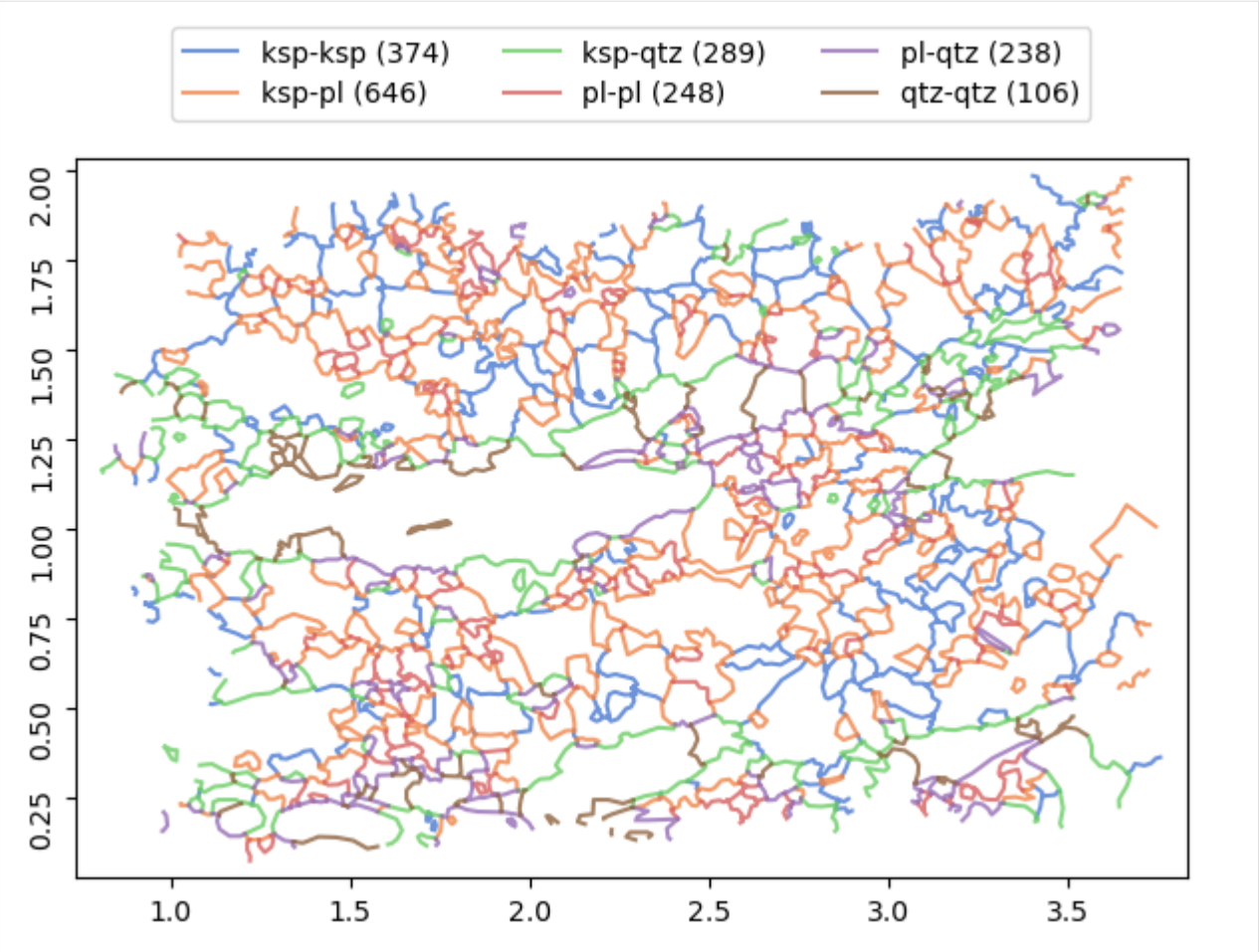


2.2 Boundaries objects

The map of grain boundaries could be created from topologically correct grains using `boundaries` method:

[18]:
```
b = g.boundaries()
```

[19]:
```
b.show()
```

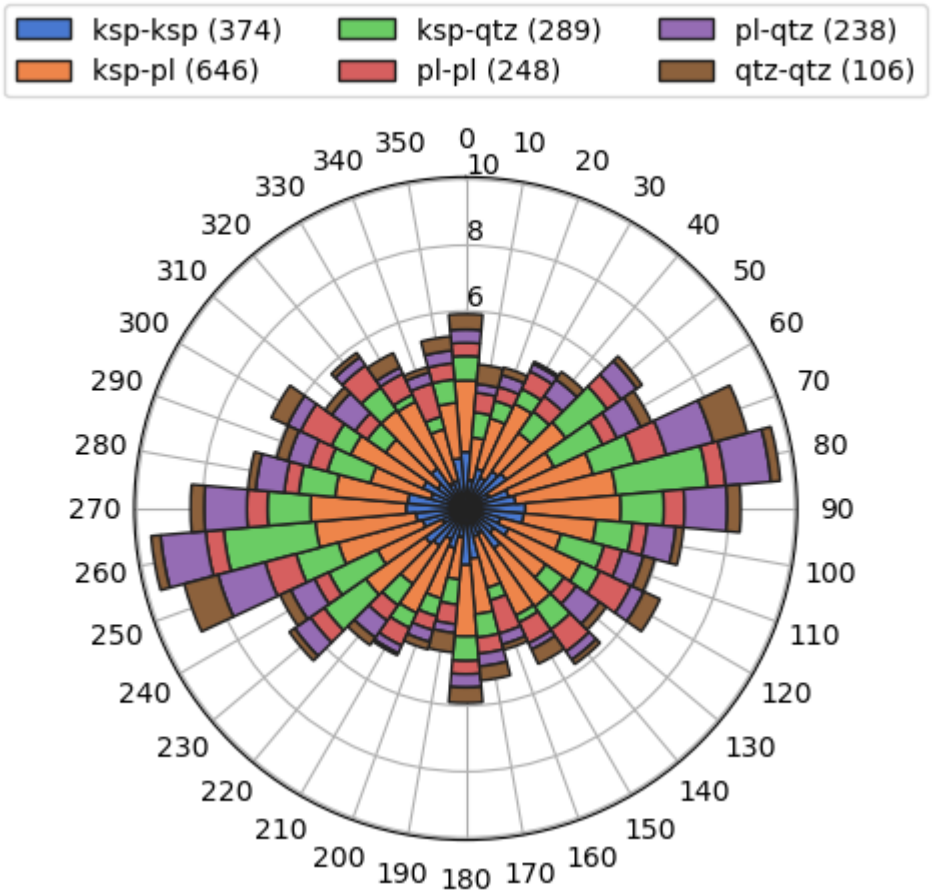


[20]:
```
df = b.groups('length').sum()
df['percents']= 100 * df['length'] / df['length'].sum()
df
```

[20]:

class	length	percents
ksp-ksp	23.383974	21.384276
ksp-pl	38.592227	35.291983
ksp-qtz	17.920424	16.387946
pl-pl	11.302490	10.335949
pl-qtz	11.535006	10.548581
qtz-qtz	6.617133	6.051264

[21]:
```
b.rose(weights=b.length, scaled=False)
```


ksp-ksp (374)
ksp-qtz (289)
pl-qtz (238)
ksp-pl (646)
pl-pl (248)
qtz-qtz (106)
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
10
8
6

[ ]:

CHAPTER

THREE

PACKAGE MODULES

PolyLX provides following modules:

3.1 core module

Python module to visualize and analyze digitized 2D microstructures.

@author: Ondrej Lexa

Examples

```
>>> from polylx import *
>>> g = Grains.example()
>>> b = g.boundaries()
```

class polylx.core.**Boundaries**(*shapes*, *classification=None*)

Bases: *PolySet*

Class to store set of `Boundaries` objects

__init__(*shapes*, *classification=None*)

accumulate(**methods*)

Returns accumulated result of multiple Group methods based on actual classification.

Example

```
>>> g.accumulate('rms_ead', 'aw_ead', 'aw_ead_log')
        rms_ead    aw_ead  aw_ead_log
class
ksp    0.110679  0.185953    0.161449
pl     0.068736  0.095300    0.086762
qtz    0.097872  0.297476    0.210481
```

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

agg(**pairs*)

Returns concatenated result of multiple aggregations (different aggregation function for different attributes) based on actual classification. For single aggregation function use directly pandas groups, e.g. g.groups('lao', 'sao').agg(circular.mean)

Example

```
>>> g.agg('area', np.sum, 'ead', np.mean, 'lao', circular.mean)
           area       ead        lao
class
ksp    2.443733  0.089710  76.875488
pl     1.083516  0.060629  94.197847
qtz    1.166097  0.068071  74.320337
```

property ar

Returns array of axial ratios

Note that axial ratio is calculated from long and short axes calculated by actual `shape method`.

property area

Return array of areas of the objects. For boundary returns 0.

barplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

bootstrap(*num=100*, *size=None*)

Bootstrap random sample generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **size** – size of bootstraped samples. Default number of objects.

Examples

```
>>> bsmean = np.mean([gs.ead.mean() for gs in g.bootstrap()])
```

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundary_segments()
```

boxplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property centroid

Returns the 2D array of geometric centers of the objects

class_iter()

property class_names

classify(**args*, ***kwargs*)

Define classification of objects.

When no aruments are provided, default unique classification based on name attribute is used.

Parameters

- **vals** – name of attribute (str) used for classification
- **values** (*or array of*) –

Keyword Arguments

- **label** – used as classification label when vals is array
- **k** – number of classes for continuous values
- **rule** – type of classification, *'unique'* for unique value mapping (for discrete values), *'equal'* for k equaly spaced bins (for continuos values), *'user'* for bins edges defined by array k (for continuous values), *'jenks'* for k fischer-jenks bins and *'quantile'* for k quantile based bins.
- **cmap** – matplotlib colormap. Default 'viridis'

Examples

```
>>> g.classify('name', rule='unique')
>>> g.classify('ar', rule='jenks', k=5)
```

clip(**bounds*)

Clip by bounds rectangle (minx, miny, maxx, maxy) tuple (float values)

clip_by_shape(*other*)

clipstrap(*num=100*, *f=0.3*)

Bootstrap random rectangular clip generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **f** – area fraction clipped from original shape. Default 0.3

Examples

```
>>> csmean = np.mean([gs.ead.mean() for gs in g.clipstrap()])
```

countplot(***kwargs*)

Plot seaborn countplot.

df(**attrs*)

Returns `pandas.DataFrame` of object attributes.

Example

```
>>> g.df('ead', 'ar')
```

property extent

Returns minimum bounding region (minx, miny, maxx, maxy) of all objects

property features

Generator of feature records

feret(*angle=0*)

Returns array of feret diameters for given angle.

Keyword Arguments
angle (*float*) – Caliper angle. Default 0

property fid

Return array of fids of objects.

classmethod from_file(*filename*, ***kwargs*)

Create Boudaries from geospatial file using fiona.

Parameters
filename – filename of geospatial file.

Keyword Arguments

- **namefield** – name of attribute that holds names of boundaries or None. Default “name”.
- **name** – value used for boundary name when namefield is None
- **layer** – name of layer in files which support it e.g. ‘GPKG’. Default boundaries

classmethod from_shp(*filename*, *namefield='name'*, *name='None'*)

Create Boundaries from ESRI shapefile.

Parameters
filename – filename of shapefile.

Keyword Arguments

- **namefield** – name of attribute in shapefile that holds names of boundairies or None. Default “name”.
- **name** – value used for grain name when namefield is None

generalize(*method='taubin'*, ***kwargs*)

get(*attr*)

Returns `pandas.Series` of object attribute.

Example

```
>>> g.get('ead')
```

get_class(*key*)

getindex(*name*)

Return the indices of the objects with given name.

gridsplit(*m=1*, *n=1*)

Rectangular split generator.

Parameters

- **m** – number of rows and columns to split.
- **n** – number of rows and columns to split.

Examples

```
>>> smean = np.mean([gs.ead.mean() for gs in g.gridsplit(6, 8)])
```

groups(**attrs*)

Returns `pandas.GroupBy` of object attributes.

Note that grouping is based on actual classification.

Example

```
>>> g.classify('ar', rule='natural')
>>> g.groups('ead').mean()
                ead
class
1.02-1.32   0.067772
1.32-1.54   0.076042
1.54-1.82   0.065479
1.82-2.37   0.073690
2.37-12.16  0.084016
```

property height

Returns height of extent.

property la

Return array of long axes of objects according to shape_method.

property lao

Return array of long axes of objects according to shape_method

property length

Return array of lengths of the objects.

property ma

Returns mean axis

Return array of mean axes calculated by actual `shape method`.

property name

Return list of names of the objects.

property names

Returns list of unique object names.

nndist(***kwargs*)

paror(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

plot(***kwargs*)

Plot set of `Grains` or `Boundaries` objects.

Keyword Arguments

- **alpha** – transparency. Default 0.8
- **pos** – legend position “top”, “right” or “none”. Defalt “auto”
- **ncol** – number of columns for legend.
- **legend** – Show legend. Default True
- **show_fid** – Show FID of objects. Default False
- **show_index** – Show index of objects. Default False
- **scalebar** – When True scalebar is drawn instead axes frame
- **scalebar_kwg** – Dict of scalebar properties size: Default 1 label: Default 1mm loc: Default ‘lower right’ See AnchoredSizeBar for others

Returns matplotlib axes object.

proj(*angle=0*)

Returns array of cumulative projection of object for given angle.

Keyword Arguments
angle (*float*) – angle of projection line. Default 0

regularize(***kwargs*)

property representative_point

Returns a 2D array of cheaply computed points that are guaranteed to be within the objects.

rose(***kwargs*)

Plot polar histogram of `Grains` or `Boundaries` orientations

Keyword Arguments

- **show** – If True matplotlib show is called. Default True
- **attr** – property used for orientation. Default ‘lao’
- **bins** – number of bins
- **weights** – if provided histogram is weighted
- **density** – True for probability density otherwise counts

- **grid** – True to show grid
- **color** – Bars color. Default is taken classification.
- **ec** – edgecolor. Default '#222222'
- **alpha** – alpha value. Default 1

When show=False, returns matplotlib axes object

rotate(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane.

The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
angle (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

property sa

Return array of long axes of objects according to shape_method

property sao

Return array of long axes of objects according to shape_method

savefig(***kwargs*)

Save grains or boudaries plot to file.

Keyword Arguments

- **filename** – file to save figure. Default "figure.png"
- **dpi** – DPI of image. Default 150
- **kwargs** (*See plot for other*) –

scale(***kwargs*)

Returns a scaled geometry, scaled by factors along each dimension.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape

Return list of shapely objects.

property shape_method

Set or returns shape methods of all objects.

show(***kwargs*)

Show of `Grains` or `Boundaries` objects.

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

swarmplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

to_file(*filename*, ***kwargs*)

Save boundaries to geospatial file

Parameters
filename – filename

Keyword Arguments

- **driver** – 'ESRI Shapefile', 'GeoJSON', 'GPKG' or 'GML'. Default 'GPKG'
- **layer** – name of layer in files which support it e.g. 'GPKG'. Default boundaries

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

violinplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property width

Returns width of extent.

class polylx.core.**Boundary**(*shape*, *name='None-None'*, *fid=0*)

Bases: *PolyShape*

Boundary class to store polyline boundary geometry

A two-dimensional linear ring.

__init__(*shape*, *name='None-None'*, *fid=0*)

Create Boundary object

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

property ar

Returns axial ratio (eccentricity)

Note that axial ratio is calculated from long and short axes calculated by actual `shape method`.

property area

Area of the shape. For boundary returns 0.

bcov()

shape_method: bcov

Short and long axes are calculated from eigenvalue analysis of geometry segments covariance matrix.

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundaries()
>>> bs1 = g[10].boundary_segments()
>>> bs2 = b[10].boundary_segments()
```

property bounds

Returns minimum bounding region (minx, miny, maxx, maxy)

catmull(***kwargs*)

Smoothing using Catmull-Rom splines

Keyword Arguments

- **alpha** (*float*) – Tension parameter 0 <= alpha <= 1 For uniform Catmull-Rom splines, alpha=0 for centripetal Catmull-Rom splines, alpha=0.5, for chordal Catmull-Rom splines, alpha=1 Default value 0.5
- **subdivs** (*int*) – Number of subdivisions of each polyline segment. Default value 10

property centroid

Returns the geometric center of the object

chaikin(***kwargs*)

Chaikin's Corner Cutting algorithm

Keyword Arguments
iters (*int*) – Number of iterations. Default value 5

Note: algorithm (roughly) doubles the amount of nodes at each iteration, therefore care should be taken when selecting the number of iterations. Instead of the original iterative algorithm by Chaikin, this implementation makes use of the equivalent multi-step algorithm introduced by Wu et al. doi: 10.1007/978-3-540-30497-5_188

chaikin2(***kwargs*)

Chaikin corner-cutting smoothing algorithm.

Keyword Arguments
iters (*int*) – Number of iterations. Default value 5

contains(*other*)

Returns True if the geometry contains the other, else False

copy()

cov()

shape_method: cov

Short and long axes are calculated from eigenvalue analysis of coordinate covariance matrix.

crosses(*other*)

Returns True if the geometries cross, else False

difference(*other*)

Returns the difference of the geometries

disjoint(*other*)

Returns True if geometries are disjoint, else False

distance(*other*)

Unitless distance to other geometry (float)

dp(***kwargs*)

Douglas–Peucker simplification.

Keyword Arguments
tolerance (*float*) – All points in the simplified object will be within the tolerance distance of the original geometry. Default Auto

equals(*other*)

Returns True if geometries are equal, else False

equals_exact(*other*, *tolerance*)

Returns True if geometries are equal to within a specified tolerance

feret(*angle=0*)

Returns the ferret diameter for given angle.

Parameters
angle – angle of caliper rotation

property hull

Returns array of vertices on convex hull of boundary geometry.

intersection(*other*)

Returns the intersection of the geometries

intersects(*other*)

Returns True if geometries intersect, else False

property length

Unitless length of the geometry (float)

property ma

Returns mean axis

Mean axis is calculated as square root of long axis multiplied by short axis. Both axes are calculated by actual `shape method`.

maxferet()

shape_method: maxferet

Long axis is defined as the maximum caliper of the polyline. Short axis correspond to caliper orthogonal to long axis. Center coordinates are set to centroid of polyline.

overlaps(*other*)

Returns True if geometries overlap, else False

paror(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- **angles** – iterable angle values. Defaut range(180)
- **normalized** – whether to normalize values. Defaut True

property pdist

Returns a cummulative along-perimeter distances.

plot(***kwargs*)

View `Boundary` geometry on figure.

proj(*angle=0*)

Returns the cumulative projection of object for given angle.

Parameters
angle – angle of projection line

regularize(***kwargs*)

Boundary vertices regularization.

Returns `Boundary` object defined by vertices regularly distributed along original `Boundary`.

Keyword Arguments

- **N** (*int*) – Number of vertices. Default 128.
- **length** (*float*) – approx. length of segments. Default None

relate(*other*)

Returns the DE-9IM intersection matrix for the two geometries (string)

property representative_point

Returns a cheaply computed point that is guaranteed to be within the object.

rotate(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane. The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
angle (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

scale(***kwargs*)

Returns a scaled geometry, scaled by factors 'xfact' and 'yfact' along each dimension. The 'origin' keyword can be 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point. Negative scale factors will mirror or reflect coordinates.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape_method

Returns shape method in use

show(***kwargs*)

Show plot of Boundary objects.

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True. The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or a coordinate tuple (x0, y0) for fixed point.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- **angles** – iterable angle values. Defaut range(180)
- **normalized** – whether to normalize values. Defaut True

symmetric_difference(*other*)

Returns the symmetric difference of the geometries (Shapely geometry)

taubin(***kwargs*)

Taubin smoothing

Keyword Arguments

- **factor** (*float*) – How far each node is moved toward the average position of its neighbours during every second iteration. 0 < factor < 1 Default value 0.5

- **mu** (*float*) – How far each node is moved opposite the direction of the average position of its neighbours during every second iteration. 0 < -mu < 1. Default value -0.5
- **steps** (*int*) – Number of smoothing steps. Default value 5

touches(*other*)

Returns True if geometries touch, else False

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

union(*other*)

Returns the union of the geometries (Shapely geometry)

vw(***kwargs*)

Visvalingam-Whyatt simplification.

The Visvalingam-Whyatt algorithm eliminates points based on their effective area. A points effective area is defined as the change in total area of the polygon by adding or removing that point.

Keyword Arguments
threshold (*float*) – Allowed total boundary length change in percents. Default 1

within(*other*)

Returns True if geometry is within the other, else False

property xy

Returns array of vertex coordinate pair.

class polylx.core.**Fractnet**(*G*, *coords=None*)

Bases: object

Class to store topological fracture networks

Properties:
G: networkx.Graph storing fracture network topology pos: a dictionary with nodes as keys and positions as values

property Cb

Average number of connections per branch

property Cl

Average number of connections per line

property Nb

Robust number of branches

property Ni

Number of isolated tips I-nodes

property Nl

Robust number of branches

property Nx

Number of crossing fractures X-nodes

property Ny

Number of fracture abutments Y-nodes

__init__(*G*, *coords=None*)

property area

Return minimum bounding rectangle area

branches_boundaries()

components()

property degree

edges_boundaries()

classmethod example()

classmethod from_boundaries(*b*)

classmethod from_file(*filename*, ***kwargs*)

Create Fractnet from geospatial file.

Parameters
filename – filename of geospatial file.

Keyword Args are passed to fiona.open()

k_order_connectivity()

Calculation of connectivity according to Zhang et al., 1992

property n_edges

property n_nodes

property node_positions

reduce()

Remove 2 degree nodes. Usefull for connectivity calculation Zhang et al., 1992

show(***kwargs*)

show_components(***kwargs*)

show_nodes(***kwargs*)

class polylx.core.Grain(*shape*, *name='None'*, *fid=0*)

Bases: *PolyShape*

Grain class to store polygonal grain geometry

A two-dimensional grain bounded by a linear ring with non-zero area. It may have one or more negative-space "holes" which are also bounded by linear rings.

Properties:
shape: `shapely.geometry.polygon.Polygon` object name: string with grain name. Default "None" fid: feature id. Default 0 shape_method: Method to calculate axes and orientation

__init__(*shape*, *name='None'*, *fid=0*)

Create `Grain` object

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

property ar

Returns axial ratio (eccentricity)

Note that axial ratio is calculated from long and short axes calculated by actual `shape method`.

property area

Area of the shape. For boundary returns 0.

bcov()

shape_method: bcov

Short and long axes are calculated from eigenvalue analysis of geometry segments covariance matrix.

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundaries()
>>> bs1 = g[10].boundary_segments()
>>> bs2 = b[10].boundary_segments()
```

property bounds

Returns minimum bounding region (minx, miny, maxx, maxy)

catmull(***kwargs*)

Smoothing using Catmull-Rom splines

Keyword Arguments

- **alpha** (*float*) – Tension parameter 0 <= alpha <= 1 For uniform Catmull-Rom splines, alpha=0 for centripetal Catmull-Rom splines, alpha=0.5, for chordal Catmull-Rom splines, alpha=1 Default value 0.5
- **subdivs** (*int*) – Number of subdivisions of each polyline segment. Default value 10

property cdir

Returns centroid-vertex directions of grain exterior

property cdist

Returns centroid-vertex distances of grain exterior

property centroid

Returns the geometric center of the object

chaikin(***kwargs*)

Chaikin's Corner Cutting algorithm

Keyword Arguments

iters (*int*) – Number of iterations. Default value 5

Note: algorithm (roughly) doubles the amount of nodes at each iteration, therefore care should be taken when selecting the number of iterations. Instead of the original iterative algorithm by Chaikin, this implementation makes use of the equivalent multi-step algorithm introduced by Wu et al. doi: 10.1007/978-3-540-30497-5_188

chaikin2(***kwargs*)

Chaikin corner-cutting smoothing algorithm.

Keyword Arguments
iters (*int*) – Number of iterations. Default value 5

property circularity

Return circularity (also called compactness) of the object. circ = length**2/ (4 * pi * area)

contains(*other*)

Returns True if the geometry contains the other, else False

copy()

cov()

shape_method: cov

Short and long axes are calculated from eigenvalue analysis of coordinate covariance matrix. Center coordinates are set to centroid of exterior.

crosses(*other*)

Returns True if the geometries cross, else False

difference(*other*)

Returns the difference of the geometries

direct()

shape_method: direct

Short, long axes and centre coordinates are calculated from direct least-square ellipse fitting. If direct fitting is not possible silently fallback to moment. Center coordinates are set to centre of fitted ellipse.

disjoint(*other*)

Returns True if geometries are disjoint, else False

distance(*other*)

Unitless distance to other geometry (float)

dp(***kwargs*)

Douglas–Peucker simplification.

Keyword Arguments
tolerance (*float*) – All points in the simplified object will be within the tolerance distance of the original geometry. Default Auto

property ead

Returns equal area diameter of grain

equals(*other*)

Returns True if geometries are equal, else False

equals_exact(*other*, *tolerance*)

Returns True if geometries are equal to within a specified tolerance

feret(*angle=0*)

Returns the ferret diameter for given angle.

Parameters
angle – angle of caliper rotation

fourier(***kwargs*)

Eliptic Fourier reconstruction.

Returns reconstructed `Grain` object using Fourier coefficients for characterizing closed contours.

Keyword Arguments

- **order** (*int*) – The order of FDC to calculate. Default 12.
- **N** (*int*) – number of vertices for reconstructed grain. Default 128.

fourier_ellipse()

shape_method: fourier_ellipse

Short and long axes are calculated from first-order approximation of contour with a Fourier series.

classmethod from_coords(*x*, *y*, *name='None'*, *fid=0*)

Create `Grain` from coordinate arrays

Example

```
>>> g=Grain.from_coords([0,0,2,2],[0,1,1,0])
>>> g.xy
array([[ 0.,  0.,  2.,  2.,  0.],
       [ 0.,  1.,  1.,  0.,  0.]])
```

property haralick

Return Haralick's circularity of the object. hcirc = mean(R) / std(R) where R is array of centroid-vertex distances

property hull

Returns array of vertices on convex hull of grain geometry.

property interiors

Returns list of arrays of vertex coordinate pair of interiors.

intersection(*other*)

Returns the intersection of the geometries

intersects(*other*)

Returns True if geometries intersect, else False

property length

Unitless length of the geometry (float)

property ma

Returns mean axis

Mean axis is calculated as square root of long axis multiplied by short axis. Both axes are calculated by actual `shape method`.

maee()

shape_method: maee

Short and long axes are calculated from minimum area enclosing ellipse. The solver is based on Khachiyan Algorithm, and the final solution is different from the optimal value by the pre-specified amount of tolerance of EAD/100.

Center coordinates are set to centre of fitted ellipse.

maxferet()

shape_method: maxferet

Long axis is defined as the maximum caliper of the polygon. Short axis correspond to caliper orthogonal to long axis. Center coordinates are set to centroid of exterior.

minbox()

shape_method: minbox

Short and long axes are claculated as width and height of smallest area enclosing box. Center coordinates are set to centre of box.

minferet()

shape_method: minferet

Short axis is defined as the minimum caliper of the polygon. Long axis correspond to caliper orthogonal to short axis. Center coordinates are set to centroid of exterior.

moment()

shape_method: moment

Short and long axes are calculated from area moments of inertia. Center coordinates are set to centroid. If moment fitting failed calculation fallback to maxferet. Center coordinates are set to centroid.

property nholes

Returns number of holes (shape interiors)

overlaps(*other*)

Returns True if geometries overlap, else False

paror(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- **angles** – iterable angle values. Defaut range(180)
- **normalized** – whether to normalize values. Defaut True

property pdist

Returns a cummulative along-perimeter distances.

plot(***kwargs*)

Plot `Grain` geometry on figure.

Note that plotted ellipse reflects actual shape method

proj(*angle=0*)

Returns the cumulative projection of object for given angle.

Parameters

angle – angle of projection line

regularize(***kwargs*)

Grain vertices regularization.

Returns `Grain` object defined by vertices regularly distributed along boundaries of original `Grain`.

Keyword Arguments

- **N** (*int*) – Number of vertices. Default 128.
- **length** (*float*) – approx. length of segments. Default None

relate(*other*)

Returns the DE-9IM intersection matrix for the two geometries (string)

property representative_point

Returns a cheaply computed point that is guaranteed to be within the object.

rotate(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane. The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
angle (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

scale(***kwargs*)

Returns a scaled geometry, scaled by factors 'xfact' and 'yfact' along each dimension. The 'origin' keyword can be 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point. Negative scale factors will mirror or reflect coordinates.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape_method

Returns shape method in use

shape_vector(***kwargs*)

Returns shape (feature) vector.

Shape (feature) vector is calculated from Fourier descriptors (FD) to index the shape. To achieve rotation invariance, phase information of the FDs are ignored and only the magnitudes FDn are used. Scale invariance is achieved by dividing the magnitudes by the DC component, i.e., FD0. Since centroid distance is a real value function, only half of the FDs are needed to index the shape.

Keyword Arguments
N (*int*) – number of vertices to regularize outline. Default 128 Note that number returned FDs is half of N.

show(***kwargs*)

Show plot of `Grain` objects.

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True. The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or a coordinate tuple (x0, y0) for fixed point.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

spline(***kwargs*)

Spline based smoothing of grains.

Keyword Arguments
densify (*int*) – factor for geometry densification. Default 5

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- **angles** – iterable angle values. Defaut range(180)
- **normalized** – whether to normalize values. Defaut True

symmetric_difference(*other*)

Returns the symmetric difference of the geometries (Shapely geometry)

taubin(***kwargs*)

Taubin smoothing

Keyword Arguments

- **factor** (*float*) – How far each node is moved toward the average position of its neighbours during every second iteration. 0 < factor < 1 Default value 0.5
- **mu** (*float*) – How far each node is moved opposite the direction of the average position of its neighbours during every second iteration. 0 < -mu < 1. Default value -0.5
- **steps** (*int*) – Number of smoothing steps. Default value 5

touches(*other*)

Returns True if geometries touch, else False

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

union(*other*)

Returns the union of the geometries (Shapely geometry)

vw(***kwargs*)

Visvalingam-Whyatt simplification.

The Visvalingam-Whyatt algorithm eliminates points based on their effective area. A points effective area is defined as the change in total area of the polygon by adding or removing that point.

Keyword Arguments
threshold (*float*) – Allowed total boundary length change in percents. Default 1

within(*other*)

Returns True if geometry is within the other, else False

property xy

Returns array of vertex coordinate pair.

Note that only vertexes from exterior boundary are returned. For interiors use interiors property.

class polylx.core.**Grains**(*shapes*, *classification=None*)

Bases: *PolySet*

Class to store set of `Grains` objects

__init__(*shapes*, *classification=None*)

accumulate(**methods*)

Returns accumulated result of multiple Group methods based on actual classification.

Example

```
>>> g.accumulate('rms_ead', 'aw_ead', 'aw_ead_log')
        rms_ead    aw_ead  aw_ead_log
class
ksp    0.110679  0.185953    0.161449
pl     0.068736  0.095300    0.086762
qtz    0.097872  0.297476    0.210481
```

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

agg(**pairs*)

Returns concatenated result of multiple aggregations (different aggregation function for different attributes) based on actual classification. For single aggregation function use directly pandas groups, e.g. g.groups('lao', 'sao').agg(circular.mean)

Example

```
>>> g.agg('area', np.sum, 'ead', np.mean, 'lao', circular.mean)
           area       ead        lao
class
ksp    2.443733  0.089710  76.875488
pl     1.083516  0.060629  94.197847
qtz    1.166097  0.068071  74.320337
```

property ar

Returns array of axial ratios

Note that axial ratio is calculated from long and short axes calculated by actual `shape method`.

property area

Return array of areas of the objects. For boundary returns 0.

areafraction_plot(***kwargs*)

property aw_ead

Returns normal area weighted mean of ead

property aw_ead_log

Returns lognormal area weighted mean of ead

barplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

bootstrap(*num=100*, *size=None*)

Bootstrap random sample generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **size** – size of bootstraped samples. Default number of objects.

Examples

```
>>> bsmean = np.mean([gs.ead.mean() for gs in g.bootstrap()])
```

boundaries(*T=None*)

Create Boundaries from Grains.

Example

```
>>> g = Grains.example()
>>> b = g.boundaries()
```

boundaries_fast(*T=None*)

Create Boundaries from Grains. Faster but not always safe implementation

Example

```
>>> g = Grains.example()
>>> b = g.boundaries_fast()
```

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundary_segments()
```

boxplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property centroid

Returns the 2D array of geometric centers of the objects

property circularity

Return array of circularities (also called compactness) of the objects. circ = length**2/area

class_iter()

property class_names

classify(**args*, ***kwargs*)

Define classification of objects.

When no aruments are provided, default unique classification based on name attribute is used.

Parameters

- **vals** – name of attribute (str) used for classification
- **values** (*or array of*) –

Keyword Arguments

- **label** – used as classification label when vals is array
- **k** – number of classes for continuous values
- **rule** – type of classification, *'unique'* for unique value mapping (for discrete values), *'equal'* for k equaly spaced bins (for continuos values), *'user'* for bins edges defined by array k (for continuous values), *'jenks'* for k fischer-jenks bins and *'quantile'* for k quantile based bins.
- **cmap** – matplotlib colormap. Default 'viridis'

Examples

```
>>> g.classify('name', rule='unique')
>>> g.classify('ar', rule='jenks', k=5)
```

clip(**bounds*)

Clip by bounds rectangle (minx, miny, maxx, maxy) tuple (float values)

clip_by_shape(*other*)

clipstrap(*num=100*, *f=0.3*)

Bootstrap random rectangular clip generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **f** – area fraction clipped from original shape. Default 0.3

Examples

```
>>> csmean = np.mean([gs.ead.mean() for gs in g.clipstrap()])
```

countplot(***kwargs*)

Plot seaborn countplot.

df(**attrs*)

Returns `pandas.DataFrame` of object attributes.

Example

```
>>> g.df('ead', 'ar')
```

property ead

Returns array of equal area diameters of grains

classmethod example()

Return example grains

property extent

Returns minimum bounding region (minx, miny, maxx, maxy) of all objects

property features

Generator of feature records

feret(*angle=0*)

Returns array of feret diameters for given angle.

Keyword Arguments
angle (*float*) – Caliper angle. Default 0

property fid

Return array of fids of objects.

classmethod from_file(*filename*, ***kwargs*)

Create Grains from geospatial file.

Parameters
filename – filename of geospatial file.

Keyword Arguments

- **namefield** – name of attribute that holds names of grains or None. Default "name".
- **name** – value used for grain name when namefield is None
- **layer** – name of layer in files which support it e.g. 'GPKG'. Default grains

classmethod from_shp(*filename*, *namefield='name'*, *name='None'*)

Create Grains from ESRI shapefile.

Parameters
filename – filename of shapefile.

Keyword Arguments

- **namefield** – name of attribute in shapefile that holds names of grains or None. Default "name".
- **name** – value used for grain name when namefield is None

generalize(*method='taubin'*, ***kwargs*)

get(*attr*)

Returns `pandas.Series` of object attribute.

Example

```
>>> g.get('ead')
```

get_class(*key*)

getindex(*name*)

Return the indices of the objects with given name.

grainsize_plot(*areaweighted=True*, ***kwargs*)

gridsplit(*m=1*, *n=1*)

Rectangular split generator.

Parameters

- **m** – number of rows and columns to split.
- **n** – number of rows and columns to split.

Examples

```
>>> smean = np.mean([gs.ead.mean() for gs in g.gridsplit(6, 8)])
```

groups(**attrs*)

Returns `pandas.GroupBy` of object attributes.

Note that grouping is based on actual classification.

Example

```
>>> g.classify('ar', rule='natural')
>>> g.groups('ead').mean()
                ead
class
1.02-1.32   0.067772
1.32-1.54   0.076042
1.54-1.82   0.065479
1.82-2.37   0.073690
2.37-12.16  0.084016
```

property haralick

Return array of Haralick's circularities of the objects. hcirc = mean(R) / std(R) where R is array of centroid-vertex distances

property height

Returns height of extent.

property la

Return array of long axes of objects according to shape_method.

property lao

Return array of long axes of objects according to shape_method

property length

Return array of lengths of the objects.

property ma

Returns mean axis

Return array of mean axes calculated by actual `shape method`.

property name

Return list of names of the objects.

property names

Returns list of unique object names.

property nholes

Returns array of number of holes (shape interiors)

nndist(***kwargs*)

paror(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

plot(***kwargs*)

Plot set of `Grains` or `Boundaries` objects.

Keyword Arguments

- **alpha** – transparency. Default 0.8
- **pos** – legend position “top”, “right” or “none”. Defalt “auto”
- **ncol** – number of columns for legend.
- **legend** – Show legend. Default True
- **show_fid** – Show FID of objects. Default False
- **show_index** – Show index of objects. Default False
- **scalebar** – When True scalebar is drawn instead axes frame
- **scalebar_kwg** – Dict of scalebar properties size: Default 1 label: Default 1mm loc: Default ‘lower right’ See AnchoredSizeBar for others

Returns matplotlib axes object.

proj(*angle=0*)

Returns array of cumulative projection of object for given angle.

Keyword Arguments
angle (*float*) – angle of projection line. Default 0

regularize(***kwargs*)

property representative_point

Returns a 2D array of cheaply computed points that are guaranteed to be within the objects.

property rms_ead

Returns root mean square of ead

rose(***kwargs*)

Plot polar histogram of `Grains` or `Boundaries` orientations

Keyword Arguments

- **show** – If True matplotlib show is called. Default True
- **attr** – property used for orientation. Default ‘lao’
- **bins** – number of bins
- **weights** – if provided histogram is weighted
- **density** – True for probability density otherwise counts
- **grid** – True to show grid
- **color** – Bars color. Default is taken classification.
- **ec** – edgecolor. Default ‘#222222’

- **alpha** – alpha value. Default 1

When show=False, returns matplotlib axes object

rotate(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane.

The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
angle (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

property sa

Return array of long axes of objects according to shape_method

property sao

Return array of long axes of objects according to shape_method

savefig(***kwargs*)

Save grains or boudaries plot to file.

Keyword Arguments

- **filename** – file to save figure. Default "figure.png"
- **dpi** – DPI of image. Default 150
- **kwargs** (*See plot for other*) –

scale(***kwargs*)

Returns a scaled geometry, scaled by factors along each dimension.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape

Return list of shapely objects.

property shape_method

Set or returns shape methods of all objects.

shape_vector(***kwargs*)

Returns array of shape (feature) vectors.

Keyword Arguments
N – number of points to regularize shape. Default 128 Routine return N/2 of FDs

show(***kwargs*)

Show of `Grains` or `Boundaries` objects.

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

swarmplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

to_file(*filename*, ***kwargs*)

Save Boundaries to geospatial file

Parameters
filename – filename of geospatial file

Keyword Arguments

- **driver** – 'ESRI Shapefile', 'GeoJSON', 'GPKG' or 'GML'. Default 'GPKG'
- **layer** – name of layer in files which support it e.g. 'GPKG'. Default grains

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

violinplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property width

Returns width of extent.

class polylx.core.**PolySet**(*shapes*, *classification=None*)

Bases: `object`

Base class to store set of `Grains` or `Boundaries` objects

Properties:
polys: list of objects extent: tuple of (xmin, ymin, xmax, ymax)

__init__(*shapes*, *classification=None*)

accumulate(**methods*)

Returns accumulated result of multiple Group methods based on actual classification.

Example

```
>>> g.accumulate('rms_ead', 'aw_ead', 'aw_ead_log')
        rms_ead    aw_ead  aw_ead_log
class
ksp    0.110679  0.185953    0.161449
pl     0.068736  0.095300    0.086762
qtz    0.097872  0.297476    0.210481
```

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

agg(**pairs*)

Returns concatenated result of multiple aggregations (different aggregation function for different attributes) based on actual classification. For single aggregation function use directly pandas groups, e.g. g.groups('lao', 'sao').agg(circular.mean)

Example

```
>>> g.agg('area', np.sum, 'ead', np.mean, 'lao', circular.mean)
           area       ead        lao
class
ksp    2.443733  0.089710  76.875488
pl     1.083516  0.060629  94.197847
qtz    1.166097  0.068071  74.320337
```

property ar

Returns array of axial ratios

Note that axial ratio is calculated from long and short axes calculated by actual `shape method`.

property area

Return array of areas of the objects. For boundary returns 0.

barplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

bootstrap(*num=100*, *size=None*)

Bootstrap random sample generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **size** – size of bootstraped samples. Default number of objects.

Examples

```
>>> bsmean = np.mean([gs.ead.mean() for gs in g.bootstrap()])
```

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundary_segments()
```

boxplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property centroid

Returns the 2D array of geometric centers of the objects

class_iter()

property class_names

classify(**args*, ***kwargs*)

Define classification of objects.

When no aruments are provided, default unique classification based on name attribute is used.

Parameters

- **vals** – name of attribute (str) used for classification
- **values** (*or array of*) –

Keyword Arguments

- **label** – used as classification label when vals is array
- **k** – number of classes for continuous values
- **rule** – type of classification, *'unique'* for unique value mapping (for discrete values), *'equal'* for k equaly spaced bins (for continuos values), *'user'* for bins edges defined by array k (for continuous values), *'jenks'* for k fischer-jenks bins and *'quantile'* for k quantile based bins.
- **cmap** – matplotlib colormap. Default 'viridis'

Examples

```
>>> g.classify('name', rule='unique')
>>> g.classify('ar', rule='jenks', k=5)
```

clip(**bounds*)

Clip by bounds rectangle (minx, miny, maxx, maxy) tuple (float values)

clip_by_shape(*other*)

clipstrap(*num=100*, *f=0.3*)

Bootstrap random rectangular clip generator.

Parameters

- **num** – number of boostraped samples. Default 100
- **f** – area fraction clipped from original shape. Default 0.3

Examples

```
>>> csmean = np.mean([gs.ead.mean() for gs in g.clipstrap()])
```

countplot(***kwargs*)

Plot seaborn countplot.

df(**attrs*)

Returns `pandas.DataFrame` of object attributes.

Example

```
>>> g.df('ead', 'ar')
```

property extent

Returns minimum bounding region (minx, miny, maxx, maxy) of all objects

property features

Generator of feature records

feret(*angle=0*)

Returns array of feret diameters for given angle.

Keyword Arguments

angle (*float*) – Caliper angle. Default 0

property fid

Return array of fids of objects.

generalize(*method='taubin'*, ***kwargs*)

get(*attr*)

Returns `pandas.Series` of object attribute.

Example

```
>>> g.get('ead')
```

get_class(*key*)

getindex(*name*)

Return the indices of the objects with given name.

gridsplit(*m=1*, *n=1*)

Rectangular split generator.

Parameters

- **m** – number of rows and columns to split.
- **n** – number of rows and columns to split.

Examples

```
>>> smean = np.mean([gs.ead.mean() for gs in g.gridsplit(6, 8)])
```

groups(**attrs*)

Returns `pandas.GroupBy` of object attributes.

Note that grouping is based on actual classification.

Example

```
>>> g.classify('ar', rule='natural')
>>> g.groups('ead').mean()
                ead
class
1.02-1.32   0.067772
1.32-1.54   0.076042
1.54-1.82   0.065479
1.82-2.37   0.073690
2.37-12.16  0.084016
```

property height

Returns height of extent.

property la

Return array of long axes of objects according to shape_method.

property lao

Return array of long axes of objects according to shape_method

property length

Return array of lengths of the objects.

property ma

Returns mean axis

Return array of mean axes calculated by actual `shape method`.

property name

Return list of names of the objects.

property names

Returns list of unique object names.

nndist(***kwargs*)

paror(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

plot(***kwargs*)

Plot set of `Grains` or `Boundaries` objects.

Keyword Arguments

- **alpha** – transparency. Default 0.8
- **pos** – legend position “top”, “right” or “none”. Defalt “auto”
- **ncol** – number of columns for legend.
- **legend** – Show legend. Default True
- **show_fid** – Show FID of objects. Default False
- **show_index** – Show index of objects. Default False
- **scalebar** – When True scalebar is drawn instead axes frame
- **scalebar_kwg** – Dict of scalebar properties size: Default 1 label: Default 1mm loc: Default ‘lower right’ See AnchoredSizeBar for others

Returns matplotlib axes object.

proj(*angle=0*)

Returns array of cumulative projection of object for given angle.

Keyword Arguments
angle (*float*) – angle of projection line. Default 0

regularize(***kwargs*)

property representative_point

Returns a 2D array of cheaply computed points that are guaranteed to be within the objects.

rose(***kwargs*)

Plot polar histogram of `Grains` or `Boundaries` orientations

Keyword Arguments

- **show** – If True matplotlib show is called. Default True
- **attr** – property used for orientation. Default ‘lao’
- **bins** – number of bins
- **weights** – if provided histogram is weighted
- **density** – True for probability density otherwise counts
- **grid** – True to show grid
- **color** – Bars color. Default is taken classification.
- **ec** – edgecolor. Default ‘#222222’
- **alpha** – alpha value. Default 1

When show=False, returns matplotlib axes object

rotate(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane.

The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
angle (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

property sa

Return array of long axes of objects according to shape_method

property sao

Return array of long axes of objects according to shape_method

savefig(***kwargs*)

Save grains or boudaries plot to file.

Keyword Arguments

- **filename** – file to save figure. Default "figure.png"
- **dpi** – DPI of image. Default 150
- **kwargs** (*See plot for other*) –

scale(***kwargs*)

Returns a scaled geometry, scaled by factors along each dimension.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape

Return list of shapely objects.

property shape_method

Set or returns shape methods of all objects.

show(***kwargs*)

Show of `Grains` or `Boundaries` objects.

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Keyword Arguments

- **angles** – iterable of angle values. Defaut range(180)
- **normalized** (*bool*) – whether to normalize values. Default True

swarmplot(*val*, ***kwargs*)

Plot seaborn swarmplot.

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

violinplot(*val*, ***kwargs*)

Plot seaborn boxplot.

property width

Returns width of extent.

class polylx.core.**PolyShape**(*shape*, *name*, *fid*)

Bases: object

Base class to store polygon or polyline

Properties:
shape: shapely.geometry object name: name of polygon or polyline. fid: feature id

Note that all properties from shapely.geometry object are inherited.

__init__(*shape*, *name*, *fid*)

affine_transform(*matrix*)

Returns a transformed geometry using an affine transformation matrix. The matrix is provided as a list or tuple with 6 items: [a, b, d, e, xoff, yoff] which defines the equations for the transformed coordinates: x' = a * x + b * y + xoff y' = d * x + e * y + yoff

property ar

Returns axial ratio (eccentricity)

Note that axial ratio is calculated from long and short axes calculated by actual shape method.

property area

Area of the shape. For boundary returns 0.

boundary_segments()

Create Boundaries from object boundary segments.

Example

```
>>> g = Grains.example()
>>> b = g.boundaries()
>>> bs1 = g[10].boundary_segments()
>>> bs2 = b[10].boundary_segments()
```

property bounds

Returns minimum bounding region (minx, miny, maxx, maxy)

catmull(***kwargs*)

Smoothing using Catmull-Rom splines

Keyword Arguments

- **alpha** (*float*) – Tension parameter 0 <= alpha <= 1 For uniform Catmull-Rom splines, alpha=0 for centripetal Catmull-Rom splines, alpha=0.5, for chordal Catmull-Rom splines, alpha=1 Default value 0.5
- **subdivs** (*int*) – Number of subdivisions of each polyline segment. Default value 10

property centroid

Returns the geometric center of the object

chaikin(***kwargs*)

Chaikin's Corner Cutting algorithm

Keyword Arguments

iters (*int*) – Number of iterations. Default value 5

Note: algorithm (roughly) doubles the amount of nodes at each iteration, therefore care should be taken when selecting the number of iterations. Instead of the original iterative algorithm by Chaikin, this implementation makes use of the equivalent multi-step algorithm introduced by Wu et al. doi: 10.1007/978-3-540-30497-5_188

contains(*other*)

Returns True if the geometry contains the other, else False

crosses(*other*)

Returns True if the geometries cross, else False

difference(*other*)

Returns the difference of the geometries

disjoint(*other*)

Returns True if geometries are disjoint, else False

distance(*other*)

Unitless distance to other geometry (float)

dp(***kwargs*)

Douglas–Peucker simplification.

Keyword Arguments

tolerance (*float*) – All points in the simplified object will be within the tolerance distance of the original geometry. Default Auto

equals(*other*)

Returns True if geometries are equal, else False

`equals_exact`(*other*, *tolerance*)

Returns True if geometries are equal to within a specified tolerance

`feret`(*angle=0*)

Returns the ferret diameter for given angle.

Parameters
`angle` – angle of caliper rotation

`intersection`(*other*)

Returns the intersection of the geometries

`intersects`(*other*)

Returns True if geometries intersect, else False

`property length`

Unitless length of the geometry (float)

`property ma`

Returns mean axis

Mean axis is calculated as square root of long axis multiplied by short axis. Both axes are calculated by actual `shape method`.

`overlaps`(*other*)

Returns True if geometries overlap, else False

`paror`(*angles=range(0, 180)*, *normalized=True*)

Returns paror function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- `angles` – iterable angle values. Defaut range(180)
- `normalized` – whether to normalize values. Defaut True

`property pdist`

Returns a cummulative along-perimeter distances.

`proj`(*angle=0*)

Returns the cumulative projection of object for given angle.

Parameters
`angle` – angle of projection line

`relate`(*other*)

Returns the DE-9IM intersection matrix for the two geometries (string)

`property representative_point`

Returns a cheaply computed point that is guaranteed to be within the object.

`rotate`(*angle*, ***kwargs*)

Returns a rotated geometry on a 2D plane. The angle of rotation can be specified in either degrees (default) or radians by setting use_radians=True. Positive angles are counter-clockwise and negative are clockwise rotations.

Parameters
`angle` (*float*) – angle of rotation

Keyword Arguments

- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

scale(***kwargs*)

Returns a scaled geometry, scaled by factors 'xfact' and 'yfact' along each dimension. The 'origin' keyword can be 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point. Negative scale factors will mirror or reflect coordinates.

Keyword Arguments

- **xfact** (*float*) – Default 1.0
- **yfact** (*float*) – Default 1.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.

Negative scale factors will mirror or reflect coordinates.

property shape_method

Returns shape method in use

skew(***kwargs*)

Returns a skewed geometry, sheared by angles 'xs' along x and 'ys' along y direction. The shear angle can be specified in either degrees (default) or radians by setting use_radians=True. The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or a coordinate tuple (x0, y0) for fixed point.

Keyword Arguments

- **xs** (*float*) – Default 0.0
- **ys** (*float*) – Default 0.0
- **origin** – The point of origin can be a keyword 'center' for the object bounding box center (default), 'centroid' for the geometry's centroid, or coordinate tuple (x0, y0) for fixed point.
- **use_radians** (*bool*) – defaut False

surfor(*angles=range(0, 180)*, *normalized=True*)

Returns surfor function values. When normalized maximum value is 1 and correspond to max feret.

Parameters

- **angles** – iterable angle values. Defaut range(180)
- **normalized** – whether to normalize values. Defaut True

symmetric_difference(*other*)

Returns the symmetric difference of the geometries (Shapely geometry)

taubin(***kwargs*)

Taubin smoothing

Keyword Arguments

- **factor** (*float*) – How far each node is moved toward the average position of its neighbours during every second iteration. 0 < factor < 1 Default value 0.5
- **mu** (*float*) – How far each node is moved opposite the direction of the average position of its neighbours during every second iteration. 0 < -mu < 1. Default value -0.5
- **steps** (*int*) – Number of smoothing steps. Default value 5

touches(*other*)

Returns True if geometries touch, else False

translate(***kwargs*)

Returns a translated geometry shifted by offsets 'xoff' along x and 'yoff' along y direction.

Keyword Arguments

- **xoff** (*float*) – Default 1.0
- **yoff** (*float*) – Default 1.0

union(*other*)

Returns the union of the geometries (Shapely geometry)

within(*other*)

Returns True if geometry is within the other, else False

class polylx.core.**Sample**(*name=''*)

Bases: object

Class to store both Grains and Boundaries objects

Properties:

g: Grains object b: Boundaries.objects T: networkx.Graph storing grain topology

__init__(*name=''*)

bids(*idx*, *name=None*)

Return array of indexes of boundaries creating grain idx

If name keyword is provided only boundaries with grains of given name are returned.

dissolve()

classmethod example()

Returns example Sample

classmethod from_grains(*grains*, *name=''*)

get_cluster(*idx*, *name=None*)

Return array of indexes of clustered grains seeded from idx.

If name keyword is provided only neighbours with given name are returned.

get_clusters()

Return dictionary with lists of clusters for each name.

neighbors(*idx*, *name=None*, *inc=False*)

Returns array of indexes of neighbouring grains.

If name keyword is provided only neighbours with given name are returned.

neighbors_dist(*show=False*, *name=None*)

Return array of nearest neighbors distances.

If name keyword is provided only neighbours with given name are returned. When keyword show is True, plot is produced.

plot(***kwargs*)

Plot overlay of `Grains` and `Boundaries` of `Sample` object.

Keyword Arguments

- **alpha** – Grains transparency. Default 0.8
- **pos** – legend position “top” or “right”. Defalt Auto
- **ncol** – number of columns for legend.
- **show_fid** – Show FID of objects. Default False
- **show_index** – Show index of objects. Default False

Returns matplotlib axes object.

show(***kwargs*)

Show plot of `Sample` objects.

triplets()

3.2 plots module

polylx.plots.**grainsize_plot**(*d*, ***kwargs*)

polylx.plots.**logdist_plot**(*d*, ***kwargs*)

polylx.plots.**normdist_plot**(*d*, ***kwargs*)

polylx.plots.**paror_plot**(*ob*, ***kwargs*)

polylx.plots.**plot_kde**(*g*, ***kwargs*)

polylx.plots.**rose_plot**(*ang*, ***kwargs*)

polylx.plots.**surfor_plot**(*ob*, ***kwargs*)

CHAPTER

FOUR

CHANGES

4.1 0.5.4 (05 Mar 2024)

- shapelysmooth methods added for smoothing
- shapely and scipy upstream fixes
- jenks and quantile rules fix
- bcov shape_method added for eigenanalysis of decomposed geometry

4.2 0.5.3 (06 Mar 2023)

- upstream fix for networkX 3
- Fracnet.from_boundaries bug fixed

4.3 0.5.2 (06 Mar 2023)

- upstream fix for shapely 3
- topological analyses added to Fracnet

4.4 0.5.1 (27 May 2021)

- fourier_ellipse shape method for Grains added
- eliptic fourier smoothing for Grains added
- added grainsize plot
- added accumulate method to Grains and Boundaries
- simple fiona reader implemented (fiona must be installed)
- added kde plot

4.4.1 0.5 (29 Jan 2019)

- rose plot groupped according to classification
- get_class, class_iter methods added to Grains and Boundaries
- seaborn added to requirements
- several seaborn categorical plots are added as methods (swarmplot, boxplot, barplot, countplot)

4.5 0.4.9 (12 Dec 2017)

- getindex method of Grains and Boundaries implemented
- Grain cdist property return centroid-vertex distance function
- Grain cdir property return centroid-vertex direction function
- Grain shape_vector property returns normalized Fourier descriptors
- Grain regularize method returns Grain with regularly distributed vertices
- Classification could be based on properties or any other values
- boundary_segments method added
- Smoothing, simplification and regularization of boundaries implemented
- Colortable for legend is persistant trough indexing. Classify method could be used to change it
- Default color table is seaborn muted for unique classification and matplotlib viridis for continuous classes

4.6 0.4.8 (04 Mar 2017)

- bugfix

4.7 0.4.6 (04 Mar 2017)

- added plots module (initial)
- representative_point for Grains implemented
- moments calculation including holes
- surfor and parror functions added
- orientation of polygons is unified and checked
- minbox shape method added

4.8 0.4.5 (12 Jan 2017)

- shell script ipolylx opens interactive console

4.9 0.4.4 (12 Jan 2017)

- Added MAEE (minimum area enclosing ellipse) to grain shape methods
- Removed embedded IPython and IPython requirements

4.10 0.4.3 (02 Sep 2016)

- IPython added to requirements

4.11 0.4.2 (02 Sep 2016)

- Sample has pairs property(dictionary) to map boundary id to grains id
- Sample triplets method returns list of grains id creating triple points

4.12 0.4.1 (20 Jun 2016)

- Examples added to distribution

4.12.1 0.4 (20 Jun 2016)

- Sample neighbors_dist method to calculate neighbors distances
- Grains and Boundaries nndist to calculate nearest neighbors distances
- Fancy indexing with slices fixed
- Affine transformations affine_transform, rotate, scale, skew, translate methods implemented for Grains and Boundaries
- Sample name atribute added
- Sample bids method to get boundary id's related to grain added

4.13 0.3.2 (04 Jun 2016)

- PolyShape name forced to be string
- Creation of boundaries is Grains method

4.14 0.3.1 (22 Feb 2016)

- classification is persitant trough fancy indexing
- empty classes allowed
- bootstrap method added to PolySet

4.14.1 0.2 (18 Apr 2015)

- Smooth and simplify methods for Grains implemented
- Initial documentation added
- `phase` and `type` properties renamed to `name`

4.14.2 0.1 (13 Feb 2015)

- First release

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X