Spatial EFD Documentation

Release 1.0.4

Stuart W.D. Grieve

May 08, 2018

Contents:

A pure python implementation of the elliptical Fourier analysis method described by Kuhl and Giardina (1982). This package is designed to allow the rapid analysis of spatial data stored as ESRI shapefiles, handling all of the geometric conversions. The resulting data can be written back to shapefiles to allow analysis with other spatial data or can be plotted using matplotlib.

The code is built upon the pyefd module and it is hoped that this package will allow more geoscientists to apply this technique to analyze spatial data using the elliptical Fourier descriptor technique as there is no longer a data conversion barrier to entry. This package is also more feature rich than previous implementations, providing calculations of Fourier power and spatial averaging of collections of ellipses.

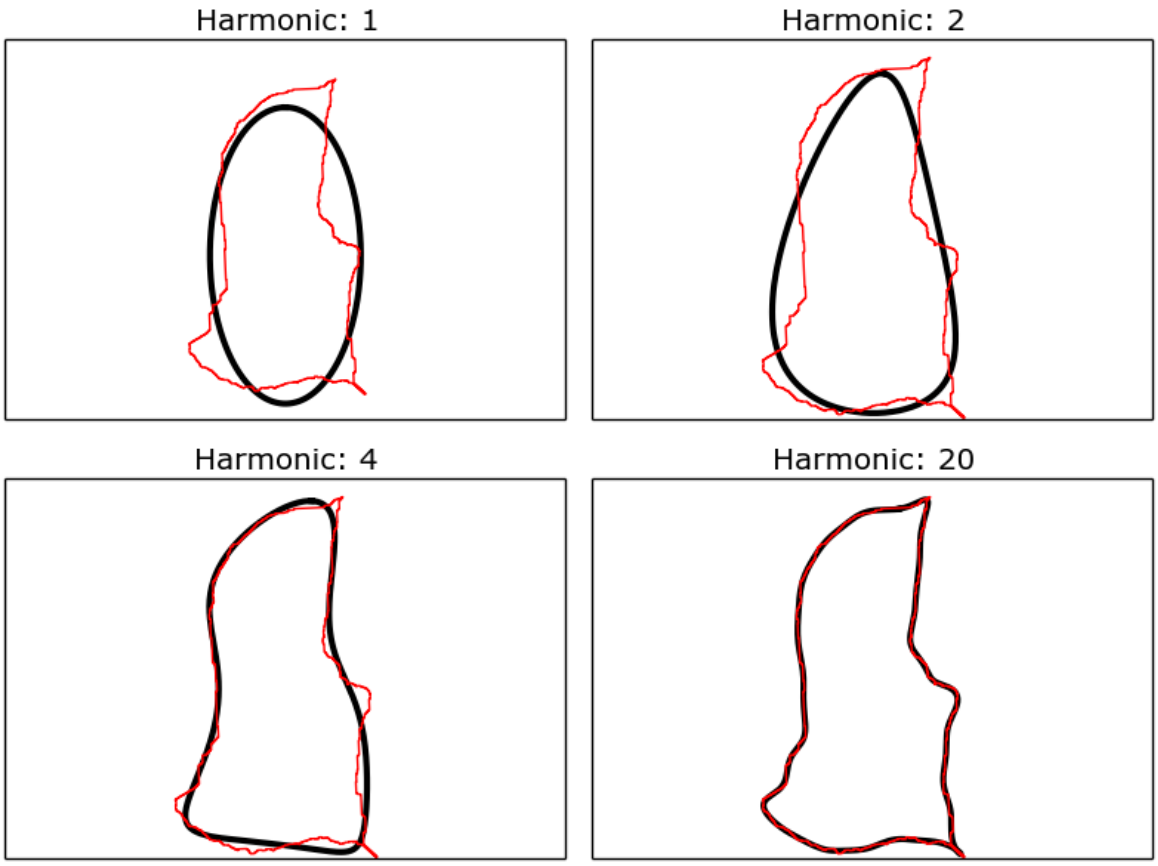

Fig. 1: Examples of Fourier ellipses (black) being fitted to a shapefile outline (red), for increasing numbers of harmonics.

CHAPTER 1

Features

- Built-in geometry processing, just pass in a shapefile and get results quickly!
- Fourier coefficient average and standard devation calculation
- Handles spatial input data through the pyshp library
- Compute an appropriate number of harmonics for a given polygon
- Basic plotting for analysis and debugging through matplotlib
- Write Fourier ellipses as shapefiles

CHAPTER 2

Installation

Install `spatial_efd` by running:

```
$ pip install spatial_efd
```

CHAPTER 3

Dependencies

This package is developed on Linux for Python 2.7 and requires `matplotlib`, `numpy`, `nose` and `pyshp`. These packages will all install automatically if `spatial_efd` is installed using `pip`.

Dependencies can be tracked by visiting requires.io

CHAPTER 4

Tests

A range of unit tests are included in the /spatial/tests/ directory. These can be run using nose:

```
$ nosetests
```

Or directly from setup.py:

```
$ python setup.py test
```

Many of these tests make use of the `example_data.shp` file which is a shapefile containing six polygons taken from a real dataset of landslide source areas.

CHAPTER 5

Usage

5.1 Normalized Data

The first step in using `spatial_efd` is always to load a shapefile:

```
import spatial_efd
shp = spatial_efd.LoadGeometries('spatial_efd/tests/example_data.shp')
```

This creates a shapefile object `shp` which contains the polygon geometries we want to analyze. As in most cases more than one polygon will be stored in an individual file, a single polygon can be selected for processing using python's list notation:

```
x, y, centroid = spatial_efd.ProcessGeometryNorm(shp[1])
```

This loads the geometry from the 2nd polygon within the shapefile into a list of x and a list of y coordinates. This method also computes the centroid of the polygon, which can be useful for later analysis. To make comparisons between data from different locations simpler, these data are normalized.

If you already know how many harmonics you wish to compute this can be specified during the calculation of the Fourier coefficients:

```
harmonic = 20
coeffs = spatial_efd.CalculateEFD(x, y, harmonic)
```

However, if you need to quantify the number of harmonics needed to exceed a threshold Fourier power, this functionality is available. To do this, an initial set of coefficients need to be computed to the number of harmonics required to equal the Nyquist frequency:

```
nyquist = spatial_efd.Nyquist(x)
tmpcoeffs = spatial_efd.CalculateEFD(x, y, nyquist)
harmonic = spatial_efd.FourierPower(tmpcoeffs, x)
coeffs = spatial_efd.CalculateEFD(x, y, harmonic)
```

Once the coefficients have been calculated they can be normalized following the steps outlined by Kuhl and Giardina (1982):

```
coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=True)
```

`size_invariant` should be set to True (the default value) in most cases to normalize the coefficient values, allowing comparison between polygons of differing sizes. Set `size_invariant` to False if it is required to plot the Fourier ellipses alongside the input shapefiles, or if the Fourier ellipses are to be written to a shapefile. These techniques which apply to normalized data are outlined later in this document.

A set of coefficients can be converted back into a series of x and y coordinates by performing an inverse transform, where the harmonic value passed in will be the harmonic reconstructed:

```
xt, yt = spatial_efd.inverse_transform(coeffs, harmonic=harmonic)
```

Wrappers around some of the basic `matplotlib` functionality is provided to speed up the visualization of results:

```
ax = spatial_efd.InitPlot()
spatial_efd.PlotEllipse(ax, xt, yt, color='k', width=1.)
spatial_efd.SavePlot(ax, harmonic, '/plots/myfigure', 'png')
```

This example generates an axis object, plots our transformed coordinates onto it with a line width of 1 and a line color of black. These axes are saved with a title denoting the harmonic used to generate the coordinates and are saved in the format provided in the location provided.

Note that as this plotting is performed using `matplotlib` many other formatting options can be applied to the created axis object, to easily create publication ready plots.

To plot an overlay of a Fourier ellipse and the original shapefile data, a convenience function has been provided to streamline the coordinate processing required. Plotting the normalized coefficients, where the data has been processed using the `ProcessGeometryNorm` method is undertaken as follows (Note that `size_invariant` has been set to `False`):

```
# size_invariant must be set to false if a normalized Fourier ellipse
# is to be plotted alongside the shapefile data
coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=False)
ax = spatial_efd.InitPlot()
spatial_efd.plotComparison(ax, coeffs, harmonic, x, y, rotation=rotation)
spatial_efd.SavePlot(ax, harmonic, '/plots/myComparison', 'png')
```

Which produces a figure like this:

All of the above examples have focused on processing a single polygon from a multipart shapefile, but in most cases multiple geometries will be required to be processed. One of the common techniques surrounding elliptical Fourier analysis is the averaging of a collection of polygons. This can be achieved as follows:

```
shp = spatial_efd.LoadGeometries('spatial_efd/tests/example_data.shp')

coeffsList = []

for shape in shp:
  x, y, centroid = spatial_efd.ProcessGeometryNorm(shape)

  harmonic = 10
  coeffs = spatial_efd.CalculateEFD(x, y, harmonic)

  coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=True)

  coeffsList.append(coeffs)
```

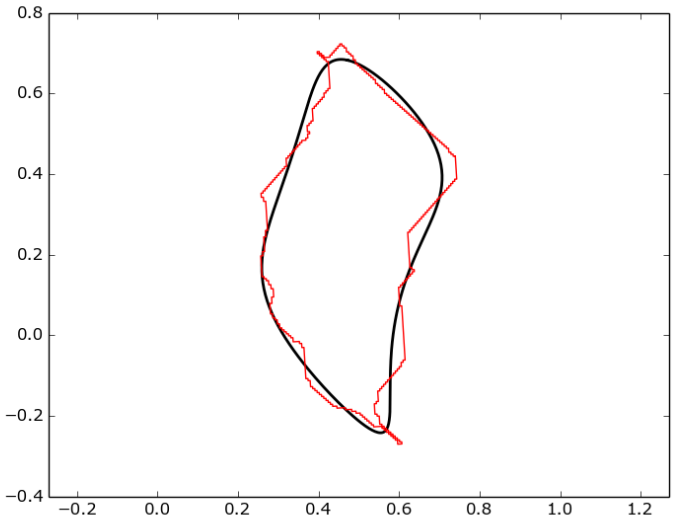

Fig. 5.1: Example of a normalized Fourier ellipse (black) being plotted on top of a shapefile outline (red).

```
avgcoeffs = spatial_efd.AverageCoefficients(coeffsList)
```

Once the average coefficients for a collection of polygons has been computed, the standard deviation can also be calculated:

```
SDcoeffs = spatial_efd.AverageSD(coeffsList, avgcoeffs)
```

With the average and standard deviation coefficients calculated, the average shape, with error ellipses can be plotted in the same manner as individual ellipses were plotted earlier

```
x_avg, y_avg = spatial_efd.inverse_transform(avgcoeffs, harmonic=harmonic)
x_sd, y_sd = spatial_efd.inverse_transform(SDcoeffs, harmonic=harmonic)

ax = spatial_efd.InitPlot()
spatial_efd.PlotEllipse(ax, x_avg, y_avg, color='b', width=2.)

# Plot avg +/- 1 SD error ellipses
spatial_efd.PlotEllipse(ax, x_avg + x_sd, y_avg + y_sd, color='k', width=1.)
spatial_efd.PlotEllipse(ax, x_avg - x_sd, y_avg - y_sd, color='k', width=1.)

spatial_efd.SavePlot(ax, harmonic, '/plots/average', 'png')
```

Which produces a figure like this:

5.2 Non-Normalized Data

In cases where the original coordinates are needed, a different processing method can be called when loading coordinates from a shapefile, to return the non-normalized data:

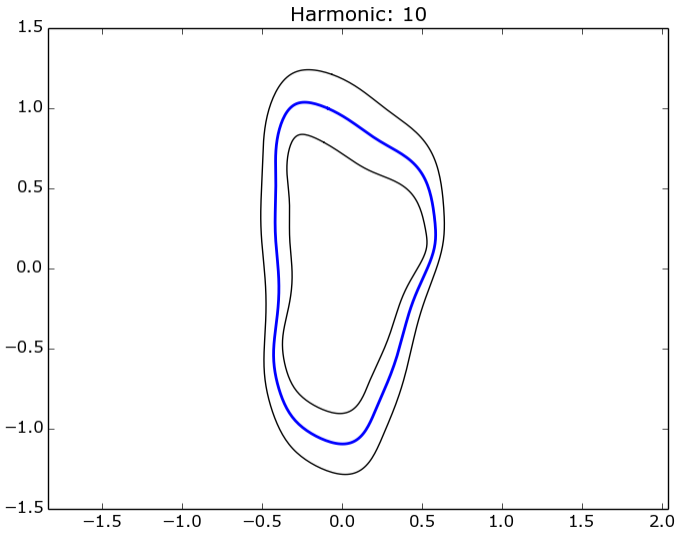

Fig. 5.2: Example of an average Fourier ellipse (blue) being plotted with standard deviation error ellipses (black).

```
x, y, centroid = spatial_efd.ProcessGeometry(shp[1])
```

This method should be used where the original coordinates need to be preserved, for example if output to a shapefile is desired. To plot non-normalized data alongside the original shapefile data, the locus of the coefficients must be computed and passed as an argument to the inverse transform method:

```
locus = spatial_efd.calculate_dc_coefficients(x, y)
xt, yt = spatial_efd.inverse_transform(coeffs, harmonic=harmonic, locus=locus)
```

To plot non-normalized coefficients, again call the `plotComparison` method, with the rotation value set to `0` as no normalization has been performed on the input data:

```
ax = spatial_efd.InitPlot()
spatial_efd.plotComparison(ax, coeffs, harmonic, x, y, rotation=0.)
spatial_efd.SavePlot(ax, harmonic, '/plots/myComparison', 'png')
```

Which produces a figure like this:

In the case of the non-normalized data plotted above, these ellipses can also be written to a shapefile to allow further analysis in a GIS package:

```
shape_id = 1
shpinstance = spatial_efd.generateShapefile()
shpinstance = spatial_efd.writeGeometry(coeffs, x, y, harmonic, shpinstance, shape_id)
spatial_efd.saveShapefile('myShapefile', shpinstance, prj='example_data.prj')
```

The first method called creates a blank shapefile object in memory, ready to be populated with Fourier ellipses. The second method can be wrapped in a loop to write as many ellipses as required to a single file. `shape_id` is written into the attribute table of the output shapefile and can be set to any integer as a means of identifying the Fourier ellipses. By passing in the existing `example.prj` file to the save method, a new projection file will be generated for the saved shapefile, ensuring that it has the correct spatial reference information for when it is loaded into a GIS package. Note that no reprojection is performed as the aim is for the input and output coordinate systems to match. If this parameter is excluded, the output shapefile will have no defined spatial reference system.

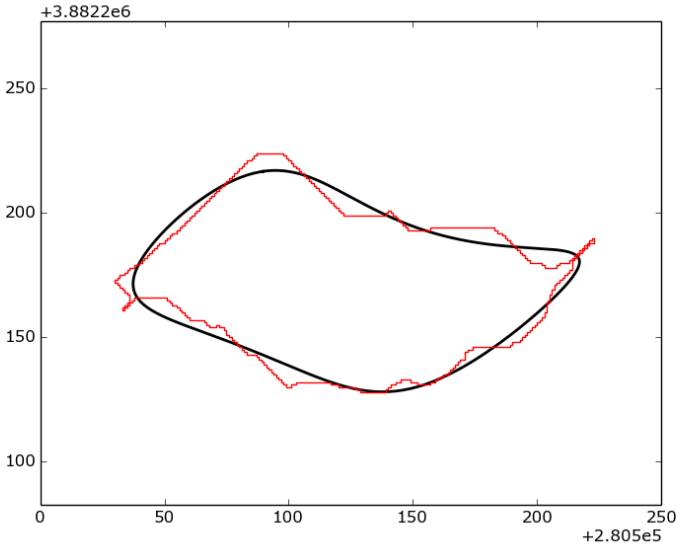

Fig. 5.3: Example of a non-normalized Fourier ellipse (black) being plotted on top of a shapefile outline (red).

For more detailed guidance on all of the functions and arguments in this package please check out the source code on github or the API documentation.

CHAPTER 6

Contribute

I welcome contributions to the code, head to the issue tracker on github to get involved!

- Issue Tracker
- Source Code

CHAPTER 7

Support

If you find any bugs, have any questions or would like to see a feature in a new version, drop me a line:

- Twitter: @GIStuart
- Email: stuart@swdg.io

CHAPTER 8

License

The project is licensed under the MIT license.

CHAPTER 9

Citation

If you use this package for scientific research please cite it as:

Grieve, S. W. D. (2017), spatial-efd: A spatial-aware implementation of elliptical Fourier analysis, The Journal of Open Source Software, 2 (11), doi:10.21105/joss.00189.

You can grab a bibtex file here.

CHAPTER 10

References

Kuhl and Giardina (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

CHAPTER 11

API

11.1 spatial_efd

spatial_efd.spatial_efd.**AverageCoefficients**(*coeffList*)
Average the coefficients contained in the list of coefficient arrays, coeffList.

This method is outlined in:

2-D particle shape averaging and comparison using Fourier descriptors: Powder Technology Volume 104, Issue 2, 1 September 1999, Pages 180-189

Parameters **coeffList** (*list*) – A list of coefficient arrays to be averaged.

Returns A numpy array containing the average An, Bn, Cn, Dn coefficient values.

Return type numpy.ndarray

spatial_efd.spatial_efd.**AverageSD**(*coeffList*, *avgcoeffs*)
Use the coefficients contained in the list of coefficient arrays, coeffList, and the average coefficient values to compute the standard deviation of series of ellipses.

This method is outlined in:

2-D particle shape averaging and comparison using Fourier descriptors: Powder Technology Volume 104, Issue 2, 1 September 1999, Pages 180-189

Parameters

- **coeffList** (*list*) – A list of coefficient arrays to be averaged.
- **avgcoeffs** (*numpy.ndarray*) – A numpy array containing the average coefficient values, generated by calling AverageCoefficients().

Returns A numpy array containing the standard deviation An, Bn, Cn, Dn coefficient values.

Return type numpy.ndarray

spatial_efd.spatial_efd.**CalculateEFD**(*X*, *Y*, *harmonics=10*)
Compute the Elliptical Fourier Descriptors for a polygon.

Implements Kuhl and Giardina method of computing the coefficients An, Bn, Cn, Dn for a specified number of harmonics. This code is adapted from the pyefd module. See the original paper for more detail:

Kuhl, FP and Giardina, CR (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.
- **harmonics** (*int*) – The number of harmonics to compute for the given shape, defaults to 10.

Returns A numpy array of shape (harmonics, 4) representing the four coefficients for each harmonic computed.

Return type numpy.ndarray

spatial_efd.spatial_efd.**CloseContour**(*X*, *Y*)

Close an opened polygon.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.

Returns A tuple containing the X and Y lists of coordinates where the first and last elements are equal.

Return type tuple

spatial_efd.spatial_efd.**ContourArea**(*X*, *Y*)

Compute the area of an irregular polygon.

Ensures the contour is closed before processing, but does not modify X or Y outside the scope of this method. Algorithm taken from http://paulbourke.net/geometry/polygonmesh/.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.

Returns The area of the input polygon.

Return type float

spatial_efd.spatial_efd.**ContourCentroid**(*X*, *Y*)

Compute the centroid of an irregular polygon.

Ensures the contour is closed before processing, but does not modify X or Y outside the scope of this method. Algorithm taken from http://paulbourke.net/geometry/polygonmesh/.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.

Returns A tuple containing the (x,y) coordinate of the center of the input polygon.

Return type tuple

spatial_efd.spatial_efd.**FourierPower**(*coeffs*, *X*, *threshold=0.9999*)

Compute the total Fourier power and find the minium number of harmonics required to exceed the threshold fraction of the total power.

This is a good method for identifying the number of harmonics to use to describe a polygon. For more details see:

3. Costa et al. / Postharvest Biology and Technology 54 (2009) 38-47

Warning: The number of coeffs must be >= the nyquist freqency.

Parameters

- **coeffs** (*numpy.ndarray*) – A numpy array of shape (n, 4) representing the four coefficients for each harmonic computed.
- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **threshold** (*float*) – The threshold fraction of the total Fourier power, the default is 0.9999.

Returns The number of harmonics required to represent the contour above the threshold Fourier power.

Return type int

spatial_efd.spatial_efd.**InitPlot**()

Set up the axes for plotting, ensuring that x and y dimensions are equal.

Returns Matplotlib axis instance.

Return type matplotlib.axes.Axes

spatial_efd.spatial_efd.**LoadGeometries**(*filename*)

Takes a filename and uses pyshp to load it, returning a list of shapefile.ShapeRecord instances.

This list can be iterated over, passing the individual shape instances to ProcessGeometry() one by one. There is no input handling if a non-polygon shapefile is passed in, that will result in undefined behavior.

Parameters **filename** (*string*) – A filename with optional full path pointing to an ESRI shapefile to be loaded by the pyshp module. The file extension is optional.

Returns A list of shapefile._ShapeRecord objects representing each polygon geometry in the shapefile.

Return type list

spatial_efd.spatial_efd.**NormContour**(*X*, *Y*, *rawCentroid*)

Normalize the coordinates which make up a contour.

Rescale the coordinates to values between 0 and 1 in both the x and y directions. The normalizing is performed using x or y width of the minimum bounding rectangle of the contour, whichever is largest. X and Y must have the same dimensions.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.
- **rawCentroid** (*tuple*) – A tuple containing the x,y coordinates of the centroid of the contour.

Returns A tuple containing a list of normalized x coordinates, a list of normalized y coordinate and the normalized centroid.

Return type tuple

spatial_efd.spatial_efd.**Nyquist**(*X*)

Returns the maximum number of harmonics that can be computed for a given contour, the nyquist freqency.

See this paper for details: C. Costa et al. / Postharvest Biology and Technology 54 (2009) 38-47

Parameters **X** (*list*) – A list (or numpy array) of x coordinate values.

Returns The nyquist frequency, expressed as a number of harmonics.

Return type int

spatial_efd.spatial_efd.**PlotEllipse**(*ax*, *x*, *y*, *color='k'*, *width=1.0*)

Plots an ellipse represented as a series of x and y coordinates on a given axis.

Parameters

- **ax** (*matplotlib.axes.Axes*) – Matplotlib axis instance.
- **x** (*list*) – A list (or numpy array) of x coordinate values.
- **y** (*list*) – A list (or numpy array) of y coordinate values.
- **color** (*string*) – A matplotlib color string to color the line used to plot the ellipse. Defaults to k (black).
- **width** (*float*) – The width of the plotted line. Defaults to 1.

spatial_efd.spatial_efd.**ProcessGeometry**(*shape*)

Method to handle all the geometry processing that may be needed by the rest of the EFD code.

Method which takes a single shape instance from a shapefile eg shp.Reader('shapefile.shp').shapeRecords()[n] where n is the index of the shape within a multipart geometry. This results in the contour, coordinate list and centroid data computed for the input polygon being normalized and returned to the user.

Parameters **shapefile._ShapeRecord** – A shapefile object representing the geometry and attributes of a single polygon from a multipart shapefile.

Returns A tuple containing a list of normalized x coordinates, a list of normalized y coordinates, contour (a list of [x,y] coordinate pairs, normalized about the shape's centroid) and the normalized coordinate centroid.

Return type tuple

spatial_efd.spatial_efd.**ProcessGeometryNorm**(*shape*)

Method to handle all the geometry processing that may be needed by the rest of the EFD code. This method normalizes the input data to allow spatially distributed data to be plotted in the same cartesian space.

Method which takes a single shape instance from a shapefile eg shp.Reader('shapefile.shp').shapeRecords()[n] where n is the index of the shape within a multipart geometry. This results in the contour, coordinate list and centroid data computed for the input polygon being normalized and returned to the user.

Parameters **shapefile._ShapeRecord** – A shapefile object representing the geometry and attributes of a single polygon from a multipart shapefile.

Returns A tuple containing a list of normalized x coordinates, a list of normalized y coordinates, contour (a list of [x,y] coordinate pairs, normalized about the shape's centroid) and the normalized coordinate centroid.

Return type tuple

spatial_efd.spatial_efd.**RotateContour**(*X*, *Y*, *rotation*, *centroid*)
Rotates a contour about a point by a given amount expressed in degrees.

Operates by calling rotatePoint() on each x,y pair in turn. X and Y must have the same dimensions.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.
- **rotation** (*float*) – The angle in degrees for the contour to be rotated by.
- **centroid** (*tuple*) – A tuple containing the x,y coordinates of the centroid to rotate the contour about.

Returns A tuple containing a list of x coordinates and a list of y coordinates.

Return type tuple

spatial_efd.spatial_efd.**SavePlot**(*ax*, *harmonic*, *filename*, *figformat='png'*)
Wrapper around the savefig method.

Call this method to add a title identifying the harmonic being plotted, and save the plot to a file. Note that harmonic is simply an int value to be appended to the plot title, it does not select a harmonic to plot.

The figformat argumet can take any value which matplotlib understands, which varies by system. To see a full list suitable for your matplotlib instance, call plt.gcf().canvas.get_supported_filetypes().

Parameters

- **ax** (*matplotlib.axes.Axes*) – Matplotlib axis instance.
- **harmonic** (*int*) – The harmonic which is being plotted.
- **filename** (*string*) – A complete path and filename, without an extension, for the saved plot.
- **figformat** (*string*) – A string denoting the format to save the figure as. Defaults to png.

spatial_efd.spatial_efd.**calculate_dc_coefficients**(*X*, *Y*)
Compute the dc coefficients, used as the locus when calling inverse_transform().

This code is adapted from the pyefd module. See the original paper for more detail:

Kuhl, FP and Giardina, CR (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

Parameters

- **X** (*list*) – A list (or numpy array) of x coordinate values.
- **Y** (*list*) – A list (or numpy array) of y coordinate values.

Returns A tuple containing the c and d coefficients.

Return type tuple

spatial_efd.spatial_efd.**generateShapefile**()
Create an empty shapefile to write output into using writeGeometry().

Builds a multipart polygon shapefile with a single attribute, ID, which can be used to reference the written polygons. Does not write any geometry or create any files.

Returns An empty polygon shapefile instance ready to have data written to it.

Return type shapefile.Writer

spatial_efd.spatial_efd.**getBBoxDimensions**(*x*, *y*)

Returns the width in the x and y dimensions and the maximum x and y coordinates for the bounding box of a given list of x and y coordinates.

Parameters

- **x** (*list*) – A list (or numpy array) of x coordinate values.
- **y** (*list*) – A list (or numpy array) of y coordinate values.

Returns A four-tuple representing (width in the x direction, width in the y direction, the minimum x coordinate and the minimum y coordinate).

Return type tuple

spatial_efd.spatial_efd.**inverse_transform**(*coeffs*, *locus=(0.0, 0.0)*, *n=300*, *harmonic=10*)

Perform an inverse fourier transform to convert the coefficients back into spatial coordinates.

Implements Kuhl and Giardina method of computing the performing the transform for a specified number of harmonics. This code is adapted from the pyefd module. See the original paper for more detail:

Kuhl, FP and Giardina, CR (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

Parameters

- **coeffs** (*numpy.ndarray*) – A numpy array of shape (n, 4) representing the four coefficients for each harmonic computed.
- **locus** (*tuple*) – The x,y coordinates of the centroid of the contour being generated. Use calculate_dc_coefficients() to generate the correct locus for a shape.
- **n** (*int*) – The number of coordinate pairs to compute. A larger value will result in a more complex shape at the expense of increased computational time. Defaults to 300.
- **harmonics** (*int*) – The number of harmonics to be used to generate coordinates, defaults to 10. Must be <= coeffs.shape[0]. Supply a smaller value to produce coordinates for a more generalized shape.

Returns A numpy array of shape (harmonics, 4) representing the four coefficients for each harmonic computed.

Return type numpy.ndarray

spatial_efd.spatial_efd.**normalize_efd**(*coeffs*, *size_invariant=True*)

Normalize the Elliptical Fourier Descriptor coefficients for a polygon.

Implements Kuhl and Giardina method of normalizing the coefficients An, Bn, Cn, Dn. Performs 3 separate normalizations. First, it makes the data location invariant by re-scaling the data to a common origin. Secondly, the data is rotated with respect to the major axis. Thirdly, the coefficients are normalized with regard to the absolute value of A_1. This code is adapted from the pyefd module. See the original paper for more detail:

Kuhl, FP and Giardina, CR (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

Parameters

- **coeffs** (*numpy.ndarray*) – A numpy array of shape (n, 4) representing the four coefficients for each harmonic computed.
- **size_invariant** (*bool*) – Set to True (the default) to perform the third normalization and false to return the data withot this processing step. Set this to False when plotting a comparison between the input data and the Fourier ellipse.

Returns

A tuple consisting of a numpy.ndarray of shape (harmonics, 4) representing the four coefficients for each harmonic computed and the rotation in degrees applied to the normalized contour.

Return type tuple

spatial_efd.spatial_efd.**plotComparison**(*ax*, *coeffs*, *harmonic*, *x*, *y*, *rotation=0.0*, *color1='k'*, *width1=2.0*, *color2='r'*, *width2=1.0*)

Convenience function which plots an EFD ellipse and a shapefile polygon in the same coordate system.

Warning: If passing in normalized coefficients, they must be created with the size_invariant parameter set to False.

Parameters

- **ax** (*matplotlib.axes.Axes*) – Matplotlib axis instance.
- **x** (*list*) – A list (or numpy array) of x coordinate values.
- **y** (*list*) – A list (or numpy array) of y coordinate values.
- **rotation** (*float*) – The angle in degrees for the contour to be rotated by. Generated by normalize_efd(). Leave as 0 if non-normalized coefficients are being plotted.
- **harmonic** (*int*) – The number of harmonics to be used to generate coordinates. Must be <= coeffs.shape[0]. Supply a smaller value to produce coordinates for a more generalized shape.
- **color1** (*string*) – A matplotlib color string to color the line used to plot the Fourier ellipse. Defaults to k (black).
- **width1** (*float*) – The width of the plotted fourier ellipse. Defaults to 1.
- **color2** (*string*) – A matplotlib color string to color the line used to plot the shapefile. Defaults to r (red).
- **width2** (*float*) – The width of the plotted shapefile. Defaults to 1.

spatial_efd.spatial_efd.**rotatePoint**(*point*, *centerPoint*, *angle*)

Rotates a point counter-clockwise around centerPoint.

The angle to rotate by is supplied in degrees. Code based on: https://gist.github.com/somada141/d81a05f172bb2df26a2c

Parameters

- **point** (*tuple*) – The point to be rotated, represented as an (x,y) tuple.
- **centerPoint** (*tuple*) – The point to be rotated about, represented as an (x,y) tuple.
- **angle** (*float*) – The angle to rotate point by, in the counter-clockwise direction.

Returns A tuple representing the rotated point, (x,y).

Return type tuple

spatial_efd.spatial_efd.**saveShapefile**(*filename*, *shpinstance*, *prj=None*)

Write the data processed by writeGeometry into a shapefile generated by generateShapefile().

If the filename of a shapfile is passed in as template, the spatial projection of the template shapefile will be applied to the new shapefile. Note that this does not carry out any coordinate transforms, it is merely to apply the same spatial reference to the output data as was present in the input data.

Parameters

- **filename** (*string*) – A complete path and filename, with or without the .shp extenion, to write the final shapefile data to. Must be a path which exists.
- **shape** (*shapefile.Writer*) – A shapefile object representing the geometry and attributes written by writeGeometry().
- **prj** (*string*) – A complete path and filename, with or without the .shp extenion, to the shapefile that data was loaded from initially, Used to copy the spatial projection information to the new file.

Warning: Code does not test if output paths exist, and if files exist they will be overwritten.

spatial_efd.spatial_efd.**writeGeometry**(*coeffs*, *x*, *y*, *harmonic*, *shpinstance*, *ID*)

Write the results of inverse_transform() to a shapefile.

Will only produce spatially meaningful data if the input coefficients have not been normalized.

Parameters

- **coeffs** (*numpy.ndarray*) – A numpy array of shape (n, 4) representing the four coefficients for each harmonic computed.
- **x** (*list*) – A list (or numpy array) of x coordinate values.
- **y** (*list*) – A list (or numpy array) of y coordinate values.
- **harmonic** (*int*) – The number of harmonics to be used to generate coordinates. Must be <= coeffs.shape[0]. Supply a smaller value to produce coordinates for a more generalized shape.
- **shpinstance** (*shapefile.Writer*) – A multipart polygon shapefile to write the data to.
- **ID** (*int*) – An integer ID value which will be written as an attribute alongside the geometry.

Returns shpinstance with the new geometry appended.

CHAPTER 12

Spatial Elliptical Fourier Descriptors

A pure python implementation of the elliptical Fourier analysis method described by Kuhl and Giardina (1982). This package is designed to allow the rapid analysis of spatial data stored as ESRI shapefiles, handling all of the geometric conversions. The resulting data can be written back to shapefiles to allow analysis with other spatial data or can be plotted using matplotlib.

The code is built upon the pyefd module and it is hoped that this package will allow more geoscientists to apply this technique to analyze spatial data using the elliptical Fourier descriptor technique as there is no longer a data conversion barrier to entry. This package is also more feature rich than previous implementations, providing calculations of Fourier power and spatial averaging of collections of ellipses.

12.1 Features

- Built-in geometry processing, just pass in a shapefile and get results quickly!
- Fourier coefficient average and standard devation calculation
- Handles spatial input data through the pyshp library
- Compute an appropriate number of harmonics for a given polygon
- Basic plotting for analysis and debugging through matplotlib
- Write Fourier ellipses as shapefiles

12.2 Installation

Install `spatial_efd` by running:

```
$ pip install spatial_efd
```

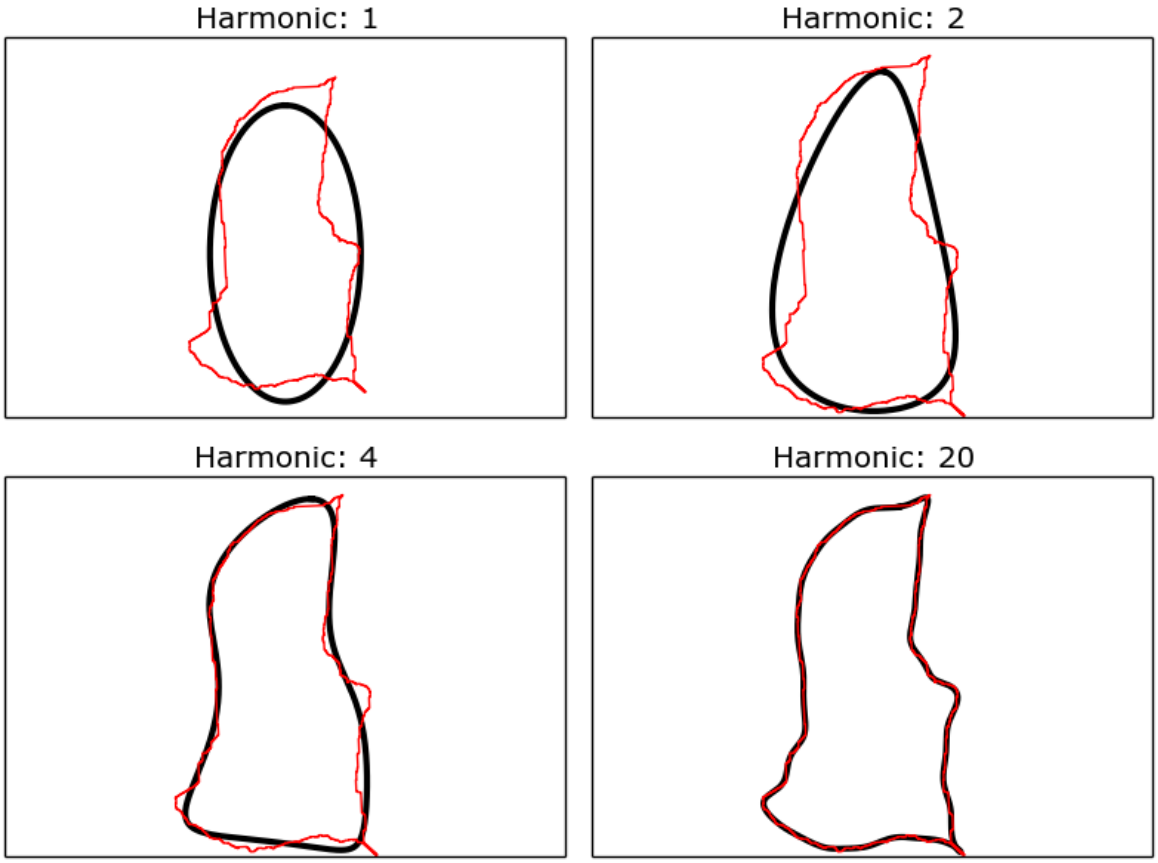

Fig. 12.1: Examples of Fourier ellipses (black) being fitted to a shapefile outline (red), for increasing numbers of harmonics.

12.3 Dependencies

This package is developed on Linux for Python 2.7 and requires `matplotlib`, `numpy`, `nose` and `pyshp`. These packages will all install automatically if `spatial_efd` is installed using `pip`.

Dependencies can be tracked by visiting requires.io

12.4 Tests

A range of unit tests are included in the /spatial/tests/ directory. These can be run using nose:

```
$ nosetests
```

Or directly from setup.py:

```
$ python setup.py test
```

Many of these tests make use of the `example_data.shp` file which is a shapefile containing six polygons taken from a real dataset of landslide source areas.

12.5 Usage

12.5.1 Normalized Data

The first step in using `spatial_efd` is always to load a shapefile:

```
import spatial_efd
shp = spatial_efd.LoadGeometries('spatial_efd/tests/example_data.shp')
```

This creates a shapefile object `shp` which contains the polygon geometries we want to analyze. As in most cases more than one polygon will be stored in an individual file, a single polygon can be selected for processing using python's list notation:

```
x, y, centroid = spatial_efd.ProcessGeometryNorm(shp[1])
```

This loads the geometry from the 2nd polygon within the shapefile into a list of x and a list of y coordinates. This method also computes the centroid of the polygon, which can be useful for later analysis. To make comparisons between data from different locations simpler, these data are normalized.

If you already know how many harmonics you wish to compute this can be specified during the calculation of the Fourier coefficients:

```
harmonic = 20
coeffs = spatial_efd.CalculateEFD(x, y, harmonic)
```

However, if you need to quantify the number of harmonics needed to exceed a threshold Fourier power, this functionality is available. To do this, an initial set of coefficients need to be computed to the number of harmonics required to equal the Nyquist frequency:

```
nyquist = spatial_efd.Nyquist(x)
tmpcoeffs = spatial_efd.CalculateEFD(x, y, nyquist)
harmonic = spatial_efd.FourierPower(tmpcoeffs, x)
coeffs = spatial_efd.CalculateEFD(x, y, harmonic)
```

Once the coefficients have been calculated they can be normalized following the steps outlined by Kuhl and Giardina (1982):

```
coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=True)
```

`size_invariant` should be set to True (the default value) in most cases to normalize the coefficient values, allowing comparison between polygons of differing sizes. Set `size_invariant` to False if it is required to plot the Fourier ellipses alongside the input shapefiles, or if the Fourier ellipses are to be written to a shapefile. These techniques which apply to normalized data are outlined later in this document.

A set of coefficients can be converted back into a series of x and y coordinates by performing an inverse transform, where the harmonic value passed in will be the harmonic reconstructed:

```
xt, yt = spatial_efd.inverse_transform(coeffs, harmonic=harmonic)
```

Wrappers around some of the basic `matplotlib` functionality is provided to speed up the visualization of results:

```
ax = spatial_efd.InitPlot()
spatial_efd.PlotEllipse(ax, xt, yt, color='k', width=1.)
spatial_efd.SavePlot(ax, harmonic, '/plots/myfigure', 'png')
```

This example generates an axis object, plots our transformed coordinates onto it with a line width of 1 and a line color of black. These axes are saved with a title denoting the harmonic used to generate the coordinates and are saved in the format provided in the location provided.

Note that as this plotting is performed using `matplotlib` many other formatting options can be applied to the created axis object, to easily create publication ready plots.

To plot an overlay of a Fourier ellipse and the original shapefile data, a convenience function has been provided to streamline the coordinate processing required. Plotting the normalized coefficients, where the data has been processed using the `ProcessGeometryNorm` method is undertaken as follows (Note that `size_invariant` has been set to `False`):

```
# size_invariant must be set to false if a normalized Fourier ellipse
# is to be plotted alongside the shapefile data
coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=False)
ax = spatial_efd.InitPlot()
spatial_efd.plotComparison(ax, coeffs, harmonic, x, y, rotation=rotation)
spatial_efd.SavePlot(ax, harmonic, '/plots/myComparison', 'png')
```

Which produces a figure like this:

All of the above examples have focused on processing a single polygon from a multipart shapefile, but in most cases multiple geometries will be required to be processed. One of the common techniques surrounding elliptical Fourier analysis is the averaging of a collection of polygons. This can be achieved as follows:

```
shp = spatial_efd.LoadGeometries('spatial_efd/tests/example_data.shp')

coeffsList = []

for shape in shp:
  x, y, centroid = spatial_efd.ProcessGeometryNorm(shape)

  harmonic = 10
  coeffs = spatial_efd.CalculateEFD(x, y, harmonic)

  coeffs, rotation = spatial_efd.normalize_efd(coeffs, size_invariant=True)

  coeffsList.append(coeffs)
```

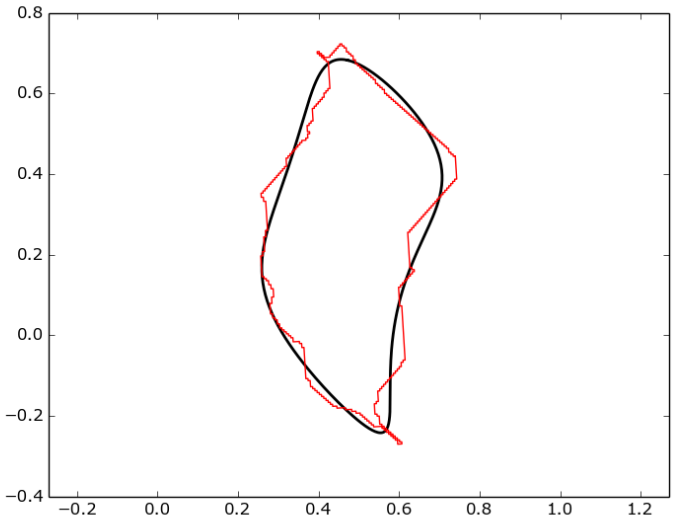

Fig. 12.2: Example of a normalized Fourier ellipse (black) being plotted on top of a shapefile outline (red).

```
avgcoeffs = spatial_efd.AverageCoefficients(coeffsList)
```

Once the average coefficients for a collection of polygons has been computed, the standard deviation can also be calculated:

```
SDcoeffs = spatial_efd.AverageSD(coeffsList, avgcoeffs)
```

With the average and standard deviation coefficients calculated, the average shape, with error ellipses can be plotted in the same manner as individual ellipses were plotted earlier

```
x_avg, y_avg = spatial_efd.inverse_transform(avgcoeffs, harmonic=harmonic)
x_sd, y_sd = spatial_efd.inverse_transform(SDcoeffs, harmonic=harmonic)

ax = spatial_efd.InitPlot()
spatial_efd.PlotEllipse(ax, x_avg, y_avg, color='b', width=2.)

# Plot avg +/- 1 SD error ellipses
spatial_efd.PlotEllipse(ax, x_avg + x_sd, y_avg + y_sd, color='k', width=1.)
spatial_efd.PlotEllipse(ax, x_avg - x_sd, y_avg - y_sd, color='k', width=1.)

spatial_efd.SavePlot(ax, harmonic, '/plots/average', 'png')
```

Which produces a figure like this:

12.5.2 Non-Normalized Data

In cases where the original coordinates are needed, a different processing method can be called when loading coordinates from a shapefile, to return the non-normalized data:

```
x, y, centroid = spatial_efd.ProcessGeometry(shp[1])
```

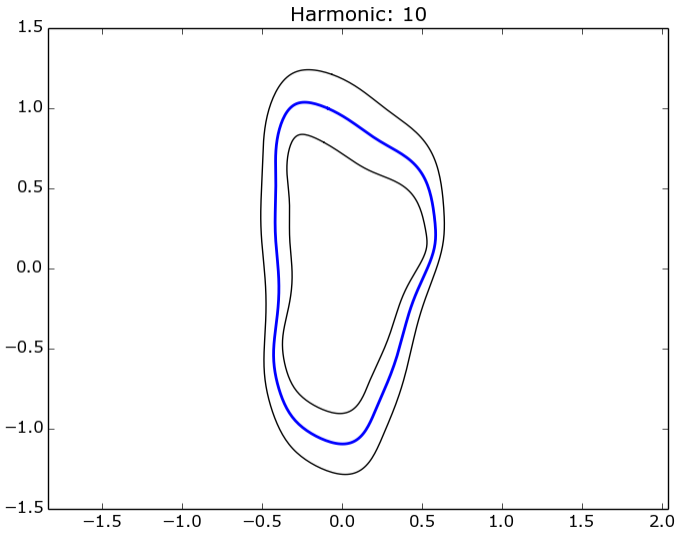

Fig. 12.3: Example of an average Fourier ellipse (blue) being plotted with standard deviation error ellipses (black).

This method should be used where the original coordinates need to be preserved, for example if output to a shapefile is desired. To plot non-normalized data alongside the original shapefile data, the locus of the coefficients must be computed and passed as an argument to the inverse transform method:

```
locus = spatial_efd.calculate_dc_coefficients(x, y)
xt, yt = spatial_efd.inverse_transform(coeffs, harmonic=harmonic, locus=locus)
```

To plot non-normalized coefficients, again call the `plotComparison` method, with the rotation value set to `0` as no normalization has been performed on the input data:

```
ax = spatial_efd.InitPlot()
spatial_efd.plotComparison(ax, coeffs, harmonic, x, y, rotation=0.)
spatial_efd.SavePlot(ax, harmonic, '/plots/myComparison', 'png')
```

Which produces a figure like this:

In the case of the non-normalized data plotted above, these ellipses can also be written to a shapefile to allow further analysis in a GIS package:

```
shape_id = 1
shpinstance = spatial_efd.generateShapefile()
shpinstance = spatial_efd.writeGeometry(coeffs, x, y, harmonic, shpinstance, shape_id)
spatial_efd.saveShapefile('myShapefile', shpinstance, prj='example_data.prj')
```

The first method called creates a blank shapefile object in memory, ready to be populated with Fourier ellipses. The second method can be wrapped in a loop to write as many ellipses as required to a single file. `shape_id` is written into the attribute table of the output shapefile and can be set to any integer as a means of identifying the Fourier ellipses. By passing in the existing `example.prj` file to the save method, a new projection file will be generated for the saved shapefile, ensuring that it has the correct spatial reference information for when it is loaded into a GIS package. Note that no reprojection is performed as the aim is for the input and output coordinate systems to match. If this parameter is excluded, the output shapefile will have no defined spatial reference system.

For more detailed guidance on all of the functions and arguments in this package please check out the source code on github or the API documentation.

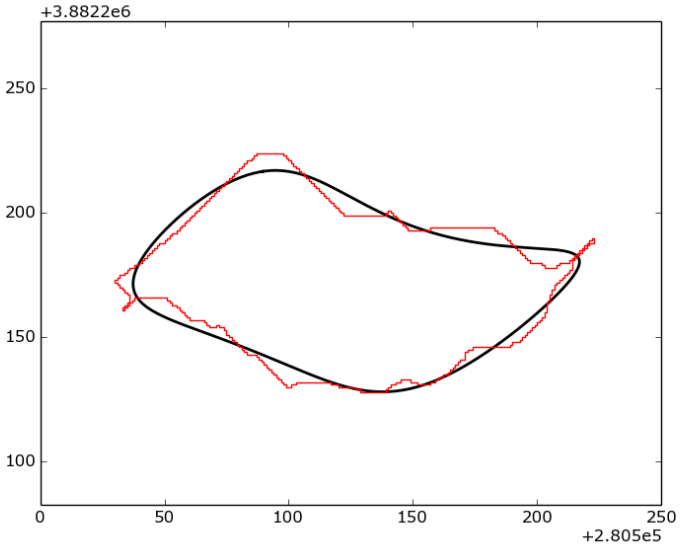

Fig. 12.4: Example of a non-normalized Fourier ellipse (black) being plotted on top of a shapefile outline (red).

12.6 Contribute

I welcome contributions to the code, head to the issue tracker on github to get involved!

- Issue Tracker
- Source Code

12.7 Support

If you find any bugs, have any questions or would like to see a feature in a new version, drop me a line:

- Twitter: @GIStuart
- Email: stuart@swdg.io

12.8 License

The project is licensed under the MIT license.

12.9 Citation

If you use this package for scientific research please cite it as:

Grieve, S. W. D. (2017), spatial-efd: A spatial-aware implementation of elliptical Fourier analysis, The Journal of Open Source Software, 2 (11), doi:10.21105/joss.00189.

You can grab a bibtex file here.

12.10 References

Kuhl and Giardina (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

12.11 API

Click here for the module level documentation.

12.12 Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index