The Journal of Open Source Software

LinkingLines: Using the Hough Transform to Cluster Line Segments and for Mesoscale Feature Extraction

Allison Kubo Hutchison [1], **Leif Karlstrom** [1], **and Tushar Mittal** [2]

1 Department of Earth Sciences, University of Oregon, Eugene, OR, USA **2** Department of Geosciences, Pennsylvania State University, University Park, PA, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Hugo Ledoux

Reviewers:

- @evetion
- @nialov

Submitted: 30 October 2023
Published: unpublished

Summary

Linear feature analysis plays a fundamental role in various scientific and geospatial applications, from detecting infrastructure networks to characterizing geological formations. In this paper, we introduce `linkinglines`, an open-source Python package tailored for the clustering, and feature extraction of linear structures in geospatial data. Our package leverages the Hough Transform, commonly used in image processing, performs clustering of line segments in the Hough Space then provides unique feature extraction methods and visualization. `linkinglines` empowers researchers, data scientists, and analysts across diverse domains to efficiently process, understand, and extract valuable insights from linear features, contributing to more informed decision-making and enhanced data-driven exploration. We have used `linkinglines` to map giant dike swarms in Kubo Hutchison et al. (2023). This JOSS paper provides an in-depth overview of the package's design, functionalities, and practical use cases, highlighting its importance as a versatile tool in geospatial data analysis.

Statement of Need

The `linkinglines` Python package addresses the need for efficient and accurate line clustering and analysis in geospatial and image data processing in addition to adding feature extraction capabilities. As the volume of data continues to grow across various domains, including remote sensing, computer vision, and geographic information systems (GIS), there is an increasing demand for tools that can simplify the extraction and analysis of linear features such as dikes, fractures, roads, rivers, and infrastructure.

The primary needs that the `linkinglines` package fulfills include:

1. **Dissected Line Extraction or Data Reduction**: In areas where land cover or data availabilty effects the complete mapping of linear features, this package can link together similarly oriented segments into lines. This is also an effective data reduction technique. The `linkinglines` package offers algorithms and utilities for identifying and extracting such features from complex data.

2. **Feature Extraction and Analysis**: The `linkinglines` package provides feature extraction capabilities, allowing users to compute essential metrics and statistics on extracted linear, radial or circumferential type features, which is valuable in various scientific and engineering applications.

3. **Geospatial and Image Data Integration**: Geospatial data often involves complex relationships between various linear features. `linkinglines` integrates seamlessly with popular geospatial libraries like `pyproj` and `matplotlib` to facilitate georeferenced data analysis and visualization.

4. **Custom Plotting and Visualization**: Effective visualization is critical for data interpretation. The package offers custom plotting scripts, making it easier to visualize results and communicate findings effectively.

5. **Cross-Domain Applicability**: The package is versatile and adaptable to multiple domains, including geospatial analysis, planetary science, and infrastructure monitoring, making it suitable for researchers, data scientists, and analysts across various disciplines.

In summary, the `linkinglines` Python package addresses a growing need for advanced line analysis and clustering tools in data-rich environments. Its capabilities empower users to efficiently process and analyze linear features in geospatial and image data, facilitating meaningful insights and supporting informed decision-making in a wide range of applications.

This package was originally developed to tackle the issue of mapped dike segments. Rugged terrain, vegetation cover, and a large area made it impossible to accurately map dikes. The length, density, and structure of the dike swarm affects how magma is transported and erupted. Scaling analysis indicated that for segments of widths of 10 m, dikes could be 10-10s of kilometers long however observed segments were two orders of magnitude lower (Morriss et al., 2020). Additionally, the complex overlapping structure of the dike swarm was difficult to analyze. We designed `linkinglines` to extract not only lines from line segments but also help analyze the mesoscale structure of the dike swarm. Using the unique properties of the Hough Transform we an extract several unique mesoscale structures within a group of lines. Our results were published in Kubo Hutchison et al. (2023) and showed an increase in dike lengths by 30x using this method and the first quantitative attempt at mapping multiscale structures in the complex dike swarms. We showed that one single radial or circumferential swarm does not fit the current data which has implications on where the magma chambers were located in the crust.

Code Structure

To use `linkinglines`, data must be in the form of a comma-separated value file with Well-Known-Text "LineString," which is a text markup language for representing vector geometry objects (13249-3, 2016). This file format can be exported from GIS software such as QGIS. Preprocessing is applied to the dataset so that only straight lines are considered in the algorithm. This is done by loading in the points from each vector object and performing linear regression, only considering those that yield a line with a $p > 0.05$. The data is then formatted into a pandas DataFrame. This package heavily uses `pandas` as the database structure for ease of use, data manipulation, and integration with `numpy` and `scipy`(McKinney, 2010).

The Journal of Open Source Software

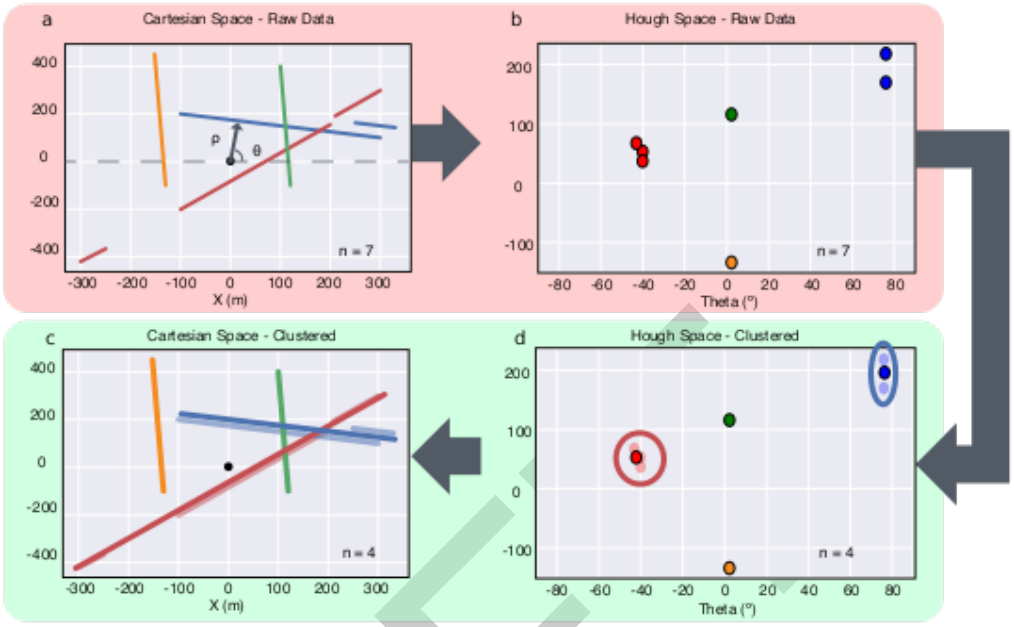

Figure 1: Dike linking algorithm using the Hough Transform. First, raw data in Cartesian space is converted into Hough space (a and b). Agglomerative clustering is then performed on the data in Hough coordinates (d). In this example, there are four dikes total and two (red and blue) clusters. The clusters are redrawn by connecting the endpoints of the segments in the cluster (c).

The Hough Transform is a image processing technique used for detecting straight lines and other patterns in binary or edge-detection images(Hough, 1962). It achieves this by converting points in an image into parametric equations and identifying patterns through the accumulation of votes in a parameter space. The transform has been generalized to detect arbitrary shapes, making it a versatile tool for pattern recognition and image analysis (Ballard, 1981). Applications of the Hough Transform include object or motion detection, lane tracking for cars, road detection in geospatial data, and biometric authentication (**survey?**). After loading in the data, it is assumed to be already line structures, so the accumulator array of the Hough Transform is skipped, although this functionality could be added if needed. First, the angle of the line segment is found using:

$$\theta = \arctan\left(\frac{-1}{m}\right) \tag{1}$$

where m is the slope of the line segment. Then the Hough Transform is performed using the following equation:

$$\rho = (x_1 - x_c)\cos\theta + (y_1 - y_c)\sin\theta \tag{2}$$

where (x_c, y_c) is the origin of the Hough Transform. In many methods, it is the left-hand corner of the image (Ballard, 1981), but we choose the average midpoint of the line segments (Figure 1B). Other origins can be specified in certain functions using the `xc` and `yc` arguments.

After the coordinate transform, ρ and θ become the basis for the Agglomerative clustering step, where we utilize Scipy's clustering algorithm (Jones et al., 2011). The clustering algorithm takes two inputs: `dtheta` and `drho`, which are used to scale the Hough Transform data. Then the clustering distance is set to 1. Combined with the default complete linkage scheme, effectively, this prevents clusters from being formed that have ranges greater than either `dtheta` or `drho` and the linear combination of the two where:

$$d = \sqrt{(\frac{\theta_1 - \theta_2}{d\theta})^2 + (\frac{\rho_1 - \rho_2}{d\rho})^2} \quad (3)$$

where two members of a potential cluster are denoted by the subscripts 1 and 2. Other linkage or distance schemes could be considered and implemented based on the specific research applications.

After labels are assigned in the clustering portion of the algorithm, new lines are drawn using the endpoints of the clustered lines (Figure 1D), which can then be output as a CSV with WKT to interface with a GIS platform. After obtaining line data and computing various statistics for each cluster, including coordinates, average rho (distance from the origin), average theta (angle), cluster size (number of lines), and other cluster-related information. For each cluster, the nearest neighbors of the segment midpoints are calculated in Cartesian space, allowing for an analysis of the Cartesian spatial clustering of the lines. We also introduce a further filtering step, which analyzes the maximum nearest neighbor difference of midpoints normalized by the total cluster length. We filter segments by setting a threshold of 0.5. This filters out clusters with segments that are not evenly clustered in Cartesian space. This step can be included or skipped in your analysis, depending on the research application using the column `TrustFilter` in the clusters DataFrame. The function returns two DataFrames: 'clusters_data,' containing summarized information for each cluster, and 'evaluation,' containing summary statistics of the clusters. Leveraging the pandas architecture allows for easy data analysis and quick referencing of the database.

Finally, we have also developed various custom plotting scripts that are helpful in investigating your clustered data.

Feature Extraction

Additionally, we leverage the unique properties of the Hough Transform to combine clustering with feature extraction. In the original usage case of overlapping complex dike swarms, two potential end members of swarm types are linear and radial or circumferential swarms (Figure 2). We can easily derive equations to describe these Cartesian patterns in the Hough Space, then perform a best-fit analysis using Jones et al. (2011).

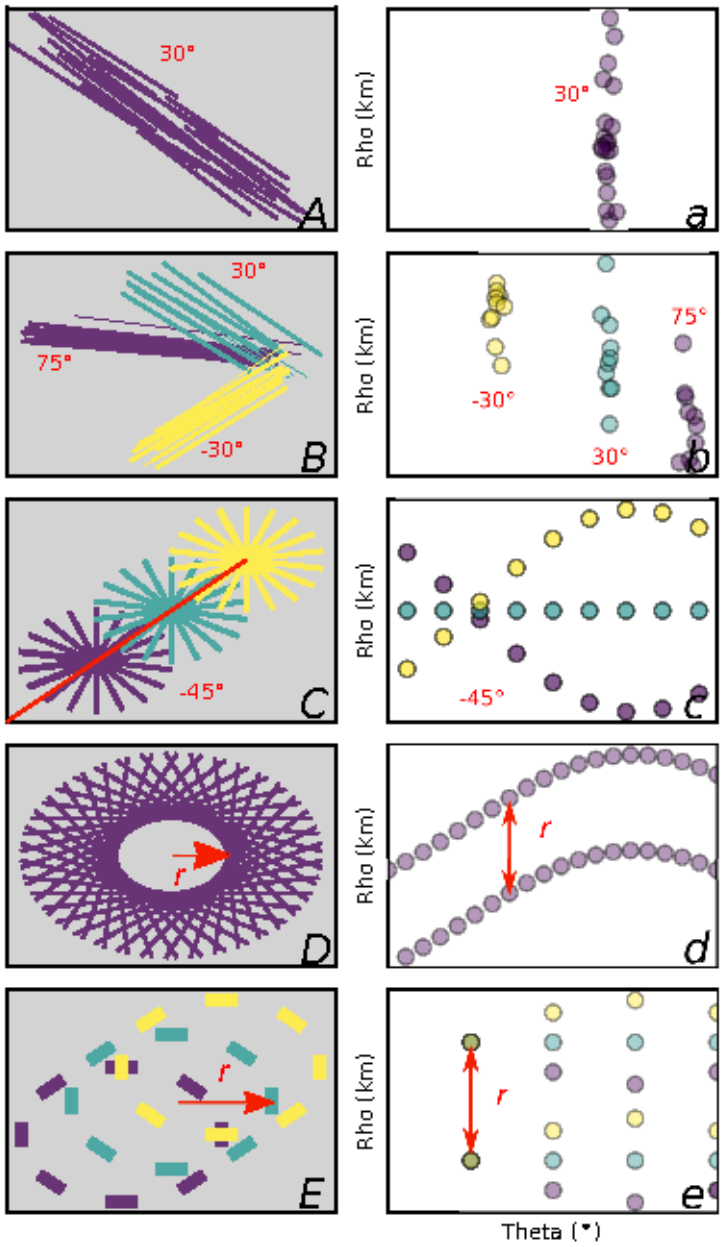

Figure 2: Synthetic dike swarms in a Cartesian space (gray background, uppercase label) and Hough Transform space (white background, lowercase label). (A) Shows a simple linear swarm oriented at 30°. (B) Shows three linear swarms at −30°, 30°, 75°. (C) Shows three radial swarms aligned at a −45° angle. The angle at which radial swarms intersect in the Hough space (HS) is the angle of their relative orientation in Cartesian space. (D) Shows a circumferential swarm with the lines extending to show how it converges to Equation 8. The radius of the circumferential swarm is equal to the spacing of the parallel two curves in HS. (E) Shows three circumferential swarms with the same radius aligned at a -45° angle.

In the case of a radial or circumferential pattern, the equation is actually the Hough Transform equation where the radial values of ρ_r can be expressed as:

$$\rho_r(\theta) = (x_r - x_c)\cos(\theta) + (x_r - y_c)\sin(\theta) \tag{4}$$

where the radial form is a function of θ and the center of the radial form, a Cartesian location (x_r, y_r) (Figure 2). Armed with this equation, we can apply a best-fit analysis of the data to find quantitative center locations for radial or circumferential patterns (Figure 2D,E). In the application of dike data, this may point to a central magma chamber or locus of stress. We can also then remove the lines which fit those patterns for feature extraction.

For extraction of linear features there are two options, one is the clustering step described above, the second can be applied with looking for mesoscale (mid scale) clusters, i.e. cluster of clusters. We apply the Hough accumulator array, a 2D histogram of θ and ρ. You can set the size of bins in the histrogram and if cluster fall within those boxes they can be thought of as mesoscale clusters. We allow for flexibility of cutoffs for these mesoscale feature extraction so it can be tailored to each research or engineering application.

Overall, these capabilities can be separated from the clustering and linking steps but is combined for ease of use in the `linkinglines` package.

Example Code Usage

After installing the code using `pip`, here is a simple usage example.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import linkinglines as ll

# Load data from a CSV file (replace 'path/to/data' with your file path)
data = pd.read_csv('path/to/data')

# Apply Hough Transform to the data to find line parameters (theta and rho)
theta, rho, xc, yc = ll.HoughTransform(data)

# Add the computed theta and rho values to the data DataFrame
data['theta'] = theta
data['rho'] = rho

# Set the increment values for clustering (adjust as needed)
dtheta = 2
drho = 500

# Perform aggregation clustering on the data
dikeset, Z = ll.AggCluster(data)

# Analyze and summarize information about the clusters
lines, evaluation = examineCluster(data)

# Create a figure and axis for visualization, coloring lines by 'AvgTheta'
fig, ax = DotsLines(lines, ColorBy='AvgTheta')
```

We have more extensive examples on our documentation site.

Future Work

This package takes geospatial or other types of line segment data and clusters them based on their orientation. Currently, the Hough transform assumes that the data input into are only

straight line segments however it could be generalized to arbitrary shapes for more flexibility (Ballard, 1981). Additionally, future work could incorporate other shapes or patterns in the Hough Space and could extend the feature extraction methods laid out here. We invite collaboration to increase the capabilities of this code.

Acknowledgements

This paper was made possible by the "Crafting Quality Research Software and Navigating Publication in Software Journals" held by the Computational Infrastructure for Geodynamics in September 2023.

References

13249-3, I. (2016). *Information technology–database languages–SQL multimedia and application packages–part 3: spatial*. International Organization for Standardization, International ….

Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, *13*(2), 111–122.

Hough, P. V. C. (1962). A method and means for recognition of complex patterns. *US Patent*, *3*, 465–467.

Jones, E., Oliphant, T., Peterson, P., & others. (2011). Scipy: Open source scientific tools for python. *Computational Science & Discovery*, *2*(1), 22.

Kubo Hutchison, A., Karlstrom, L., & Mittal, T. (2023). Multiscale spatial patterns in giant dike swarms identified through objective feature extraction. *Geochemistry, Geophysics, Geosystems*, *24*(9), e2022GC010842.

McKinney, W. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, *445*, 51–56.

Morriss, M. C., Karlstrom, L., Nasholds, M. W., & Wolff, J. A. (2020). The chief joseph dike swarm of the columbia river flood basalts, and the legacy data set of william h. taubeneck. *Geosphere*, *16*(4), 1082–1106.