**Nearest Neighbor Analysis Documentation**

* **Shubham J.**

**Overview**

This code performs a nearest neighbor analysis on spatial point patterns to compute distances and angles between nearest neighbors. It then estimates and visualizes the orientation density of these angles.

**Components**

1. **Nearest Neighbor Analysis**: Calculates the distance and angle between each point and its nearest neighbor.
2. **Orientation Density Estimation**: Estimates the density of nearest neighbor angles using a Gaussian kernel.
3. **Data Loading**: Imports spatial point data from a CSV file.
4. **Visualization**: Plots the spatial distribution of points, their nearest neighbors, and the orientation density of the nearest neighbor angles.

**Functions**

**1. nearest\_neighbor\_analysis(points)**

**Purpose**: Computes the nearest neighbor distances and angles for a given set of points.

**Parameters**:

* points (numpy.ndarray): A 2D array where each row represents the coordinates of a point (e.g., [[x1, y1], [x2, y2], ...]).

**Returns**:

* nearest\_distances (numpy.ndarray): Array of distances to the nearest neighbor for each point.
* angles (numpy.ndarray): Array of angles (in degrees) between each point and its nearest neighbor.

**Details**:

* Computes the distance matrix for all points.
* Sets the diagonal of the distance matrix to infinity to exclude self-distances.
* Finds the nearest neighbor for each point by selecting the minimum distance.
* Calculates angles using the arctan2 function and converts them to degrees.

**Code**:

python

Copy code

def nearest\_neighbor\_analysis(points):

n\_points = len(points)

dist\_matrix = distance\_matrix(points, points)

# Set diagonal to infinity to exclude self-distances

np.fill\_diagonal(dist\_matrix, np.inf)

# Find nearest neighbors

nearest\_neighbors = np.argmin(dist\_matrix, axis=1)

nearest\_distances = np.min(dist\_matrix, axis=1)

# Calculate angles in degrees (polar coordinates)

vectors = points[nearest\_neighbors] - points

angles = np.degrees(np.arctan2(vectors[:, 1], vectors[:, 0]))

return nearest\_distances, angles

**2. orientation\_density(angles, bandwidth, angle\_range=(0, 360))**

**Purpose**: Estimates the density of nearest neighbor angles using a Gaussian kernel.

**Parameters**:

* angles (numpy.ndarray): Array of angles (in degrees) between points and their nearest neighbors.
* bandwidth (float): Bandwidth for the Gaussian kernel in degrees.
* angle\_range (tuple of floats): Range of angles for which to estimate the density (default is (0, 360)).

**Returns**:

* angle\_bins (numpy.ndarray): Array of angle bin edges.
* density (numpy.ndarray): Estimated density of angles for each bin.

**Details**:

* Uses a Gaussian kernel to estimate the density function.
* The kernel function smooths the angle distribution based on the specified bandwidth.

**Code**:

python

Copy code

def orientation\_density(angles, bandwidth, angle\_range=(0, 360)):

angle\_bins = np.linspace(angle\_range[0], angle\_range[1], 360)

density = np.zeros\_like(angle\_bins)

for angle in angles:

density += np.exp(-0.5 \* ((angle\_bins - angle) / bandwidth) \*\* 2)

density /= (bandwidth \* np.sqrt(2 \* np.pi))

return angle\_bins, density

**3. load\_data\_from\_csv(file\_path)**

**Purpose**: Loads spatial point data from a CSV file.

**Parameters**:

* file\_path (str): Path to the CSV file containing the point coordinates.

**Returns**:

* points (numpy.ndarray): Array of points loaded from the CSV file.

**Details**:

* Reads the CSV file and extracts the 'X' and 'Y' coordinates as a numpy array.

**Code**:

python

Copy code

def load\_data\_from\_csv(file\_path):

data = pd.read\_csv(file\_path)

points = data[['X', 'Y']].values

return points

**Usage**

1. **Set Parameters**:
   * Define the bandwidth for density estimation.
   * Specify the path to the CSV file containing the point coordinates.
2. **Load Data**:

python

Copy code

file\_path = 'X\_nodes\_coordinates.csv'

points = load\_data\_from\_csv(file\_path)

1. **Perform Analysis**:

python

Copy code

nearest\_distances, angles = nearest\_neighbor\_analysis(points)

1. **Estimate Orientation Density**:

python

Copy code

bandwidth = 5 # Bandwidth in degrees

angle\_bins, density = orientation\_density(angles, bandwidth)

1. **Visualize Results**:

python

Copy code

plt.figure(figsize=(12, 6))

# Plot points and nearest neighbors

plt.subplot(1, 2, 1)

plt.scatter(points[:, 0], points[:, 1], c='blue', label='Points')

for i, point in enumerate(points):

nearest\_point = points[np.argmin(distance\_matrix([point], points))]

plt.plot([point[0], nearest\_point[0]], [point[1], nearest\_point[1]], 'r--', linewidth=0.5)

plt.title('Nearest Neighbors')

plt.xlabel('X coordinate')

plt.ylabel('Y coordinate')

# Plot orientation density in degrees

plt.subplot(1, 2, 2)

plt.plot(angle\_bins, density, color='red', lw=2)

plt.title('Orientation Density (Degrees)')

plt.xlabel('Angle (degrees)')

plt.ylabel('Density')

plt.show()

**Example Output**

The resulting plots will display:

* A scatter plot of the spatial points and lines connecting each point to its nearest neighbor.
* A plot of the orientation density function showing the distribution of angles between nearest neighbors.