visualization_demo

May 25, 2022

[1]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import collections as mc

import os
import time
import importlib as imp

import flow_fields as flow
import autocorrelation as ac
import spherical_harmonics as sh
for mod in [flow, ac, sh]:
    imp.reload(mod)  # Updates any customizations you've made to the module

# Set matplotlib format
SMALL_SIZE = 12
MEDIUM_SIZE = 15
BIGGER_SIZE = 18

plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

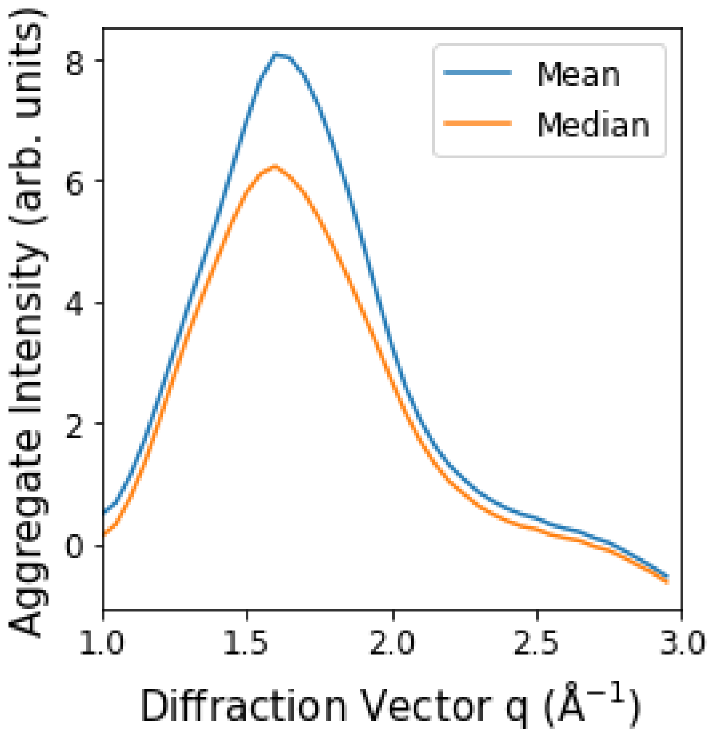
1 Load Data and Metadata

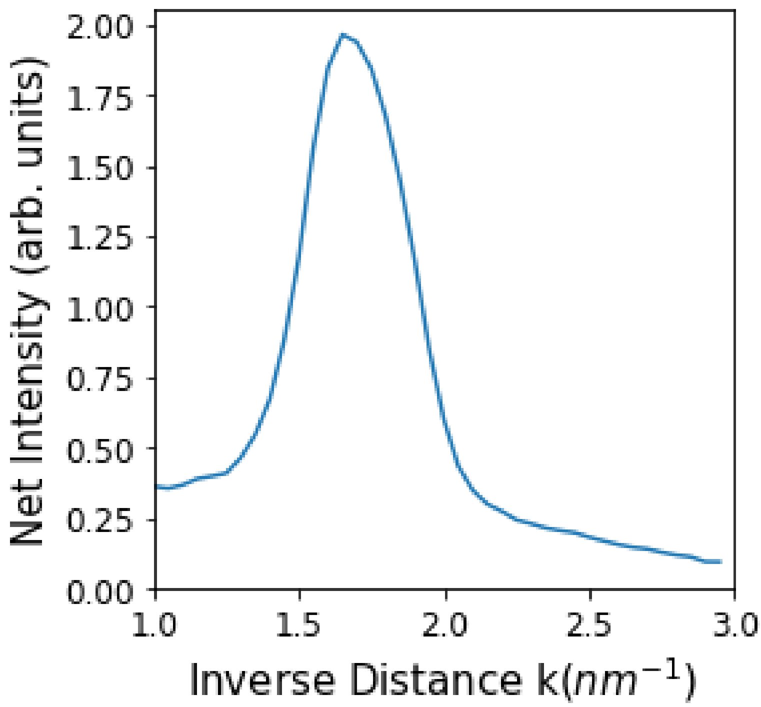
[2]:
```
# Load preprocessed 4D-STEM data and manually add metadata
path = '../data/4DSTEM/'
filename = 'sample_preprocessed_data_80x80_ss=10_cl=480__q 1.00 3.00 0.05 a 5.
↪00.npy'

q_min, q_max, q_step = (1.0, 3.0, 0.05)
```

```
angle_step = 5
step_size = 10  # nm.  This is the distance separating each diffraction pattern␣
↪in the 4D-STEM scan.

data = np.load(path + filename)
```

2 Choose Integration Range

[3]:
```
# Preview Peak Positions
aggregate_plot, difference_plot = flow.preview_intensity_vs_q(data, q_min,␣
↪q_max, q_step)
```






```
[4]: # Use the peak preview to choose the integration range for your peaks.

     # Note that there are two convoluted peaks represented in the preview, and we␣
      ↪only want one of them.

     peak_start = 1.6
     peak_end = 1.8  # Inclusive

     integrated_intensity = flow.integrate_peaks(data, q_min, q_max, q_step,␣
      ↪peak_start, peak_end)
     prepped_intensity = flow.format_flow_inputs(integrated_intensity, rotate=True)
```

3 Create Flow Plot

```
[5]: peak_width = 40  # degrees
     peaks_per_site = 1.2  # This is manually set by the user by viewing the peaks␣
      ↪present in the diffraction patterns.

     peak_positions = flow.select_peaks(prepped_intensity, peak_width,␣
      ↪peaks_per_site)
```

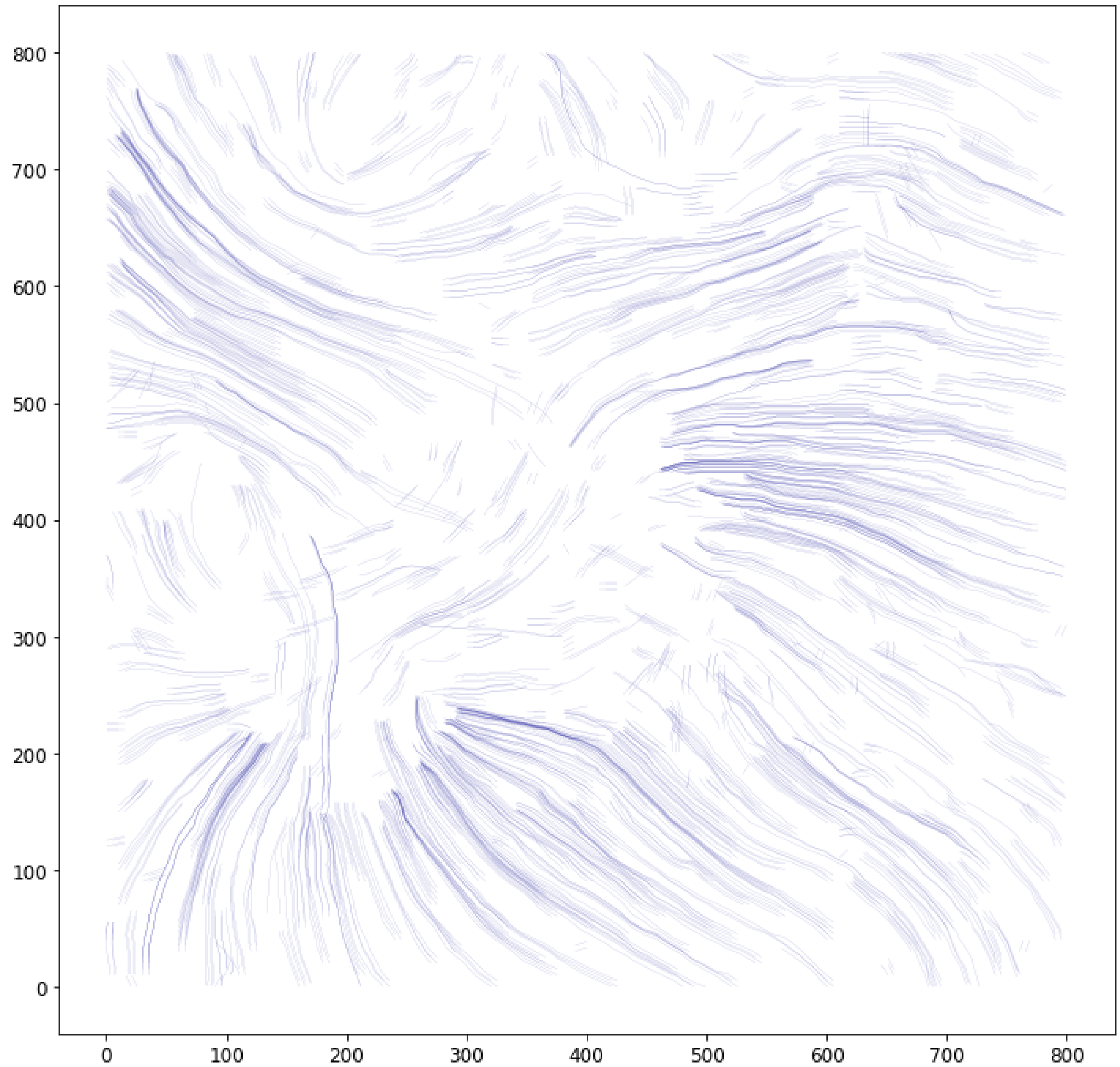
Finding Peaks…

```
Cutoff = 1.06
Peak Width = 40
7680 total peaks, which is an average of 1.20 peaks per grid square
```

[6]:
```
# Create line seeds at each peak
seed_density = 2
line_seeds = flow.seed_lines(peak_positions, step_size,␣
↪seed_density=seed_density)

# Extend line seeds to create full lines
bend_tolerance = 20
curve_resolution = 2
propagated_lines = flow.propagate_lines(line_seeds, peak_positions, step_size,␣
↪bend_tolerance,
                                    curve_resolution=curve_resolution,␣
↪max_grid_length=100)

# Show a preview, using a subset of the propagated lines
preview_sparsity = 20
propagated_image = flow.plot_solid_lines(propagated_lines, min_length=2,␣
↪sparsity=preview_sparsity)
plt.show()

# Thin out lines, reducing overlap between lines and creating a more␣
↪homogeneous line density.  This prevents the
# illusion of high density in regions with good alignment, and makes the image␣
↪more readable.
line_spacing = 1
spacing_resolution = 10
angle_spacing_degrees = 10
max_overlap_fraction = 0.5
trimmed_lines = flow.trim_lines(propagated_lines, prepped_intensity.shape,␣
↪step_size,
                                line_spacing, spacing_resolution,␣
↪angle_spacing_degrees,
                                max_overlap_fraction=max_overlap_fraction,␣
↪min_length=5, verbose=False)
trimmed_image = flow.plot_solid_lines(trimmed_lines)
plt.show()

# Add intensity data to lines
line_data = flow.prepare_line_data(trimmed_lines, prepped_intensity, step_size)
angle_data = line_data[2, :, :]
intensity_data = np.array(line_data[4, :, :])
n_dims, max_length, n_lines = line_data.shape
```

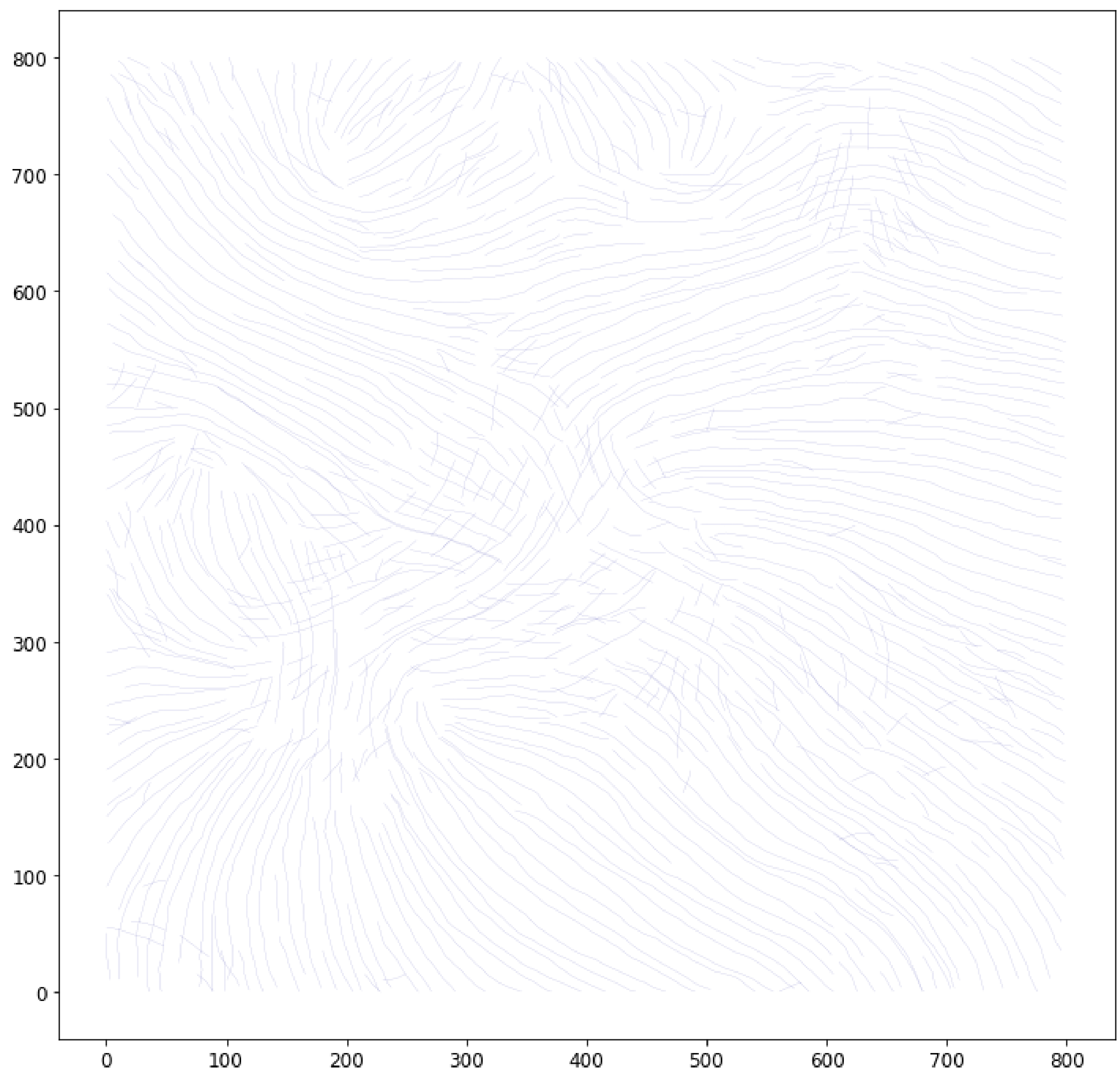
```
Seeding Lines…
```

```
Propogating Lines…
0%
5 %             0:02  remaining
10 %             0:02  remaining
15 %             0:01  remaining
20 %             0:01  remaining
25 %             0:01  remaining
30 %             0:01  remaining
35 %             0:01  remaining
40 %             0:00  remaining
45 %             0:00  remaining
50 %             0:00  remaining
55 %             0:00  remaining
60 %             0:00  remaining
65 %             0:00  remaining
70 %             0:00  remaining
75 %             0:00  remaining
80 %             0:00  remaining
85 %             0:00  remaining
90 %             0:00  remaining
95 %             0:00  remaining
Finished in  0:01
Plotting Solid Lines…
```




```
Trimming Lines…
0%
5 %            0:05  remaining
10 %            0:04  remaining
15 %            0:03  remaining
20 %            0:02  remaining
25 %            0:02  remaining
30 %            0:02  remaining
35 %            0:01  remaining
40 %            0:01  remaining
45 %            0:01  remaining
50 %            0:01  remaining
55 %            0:01  remaining
60 %            0:01  remaining
```

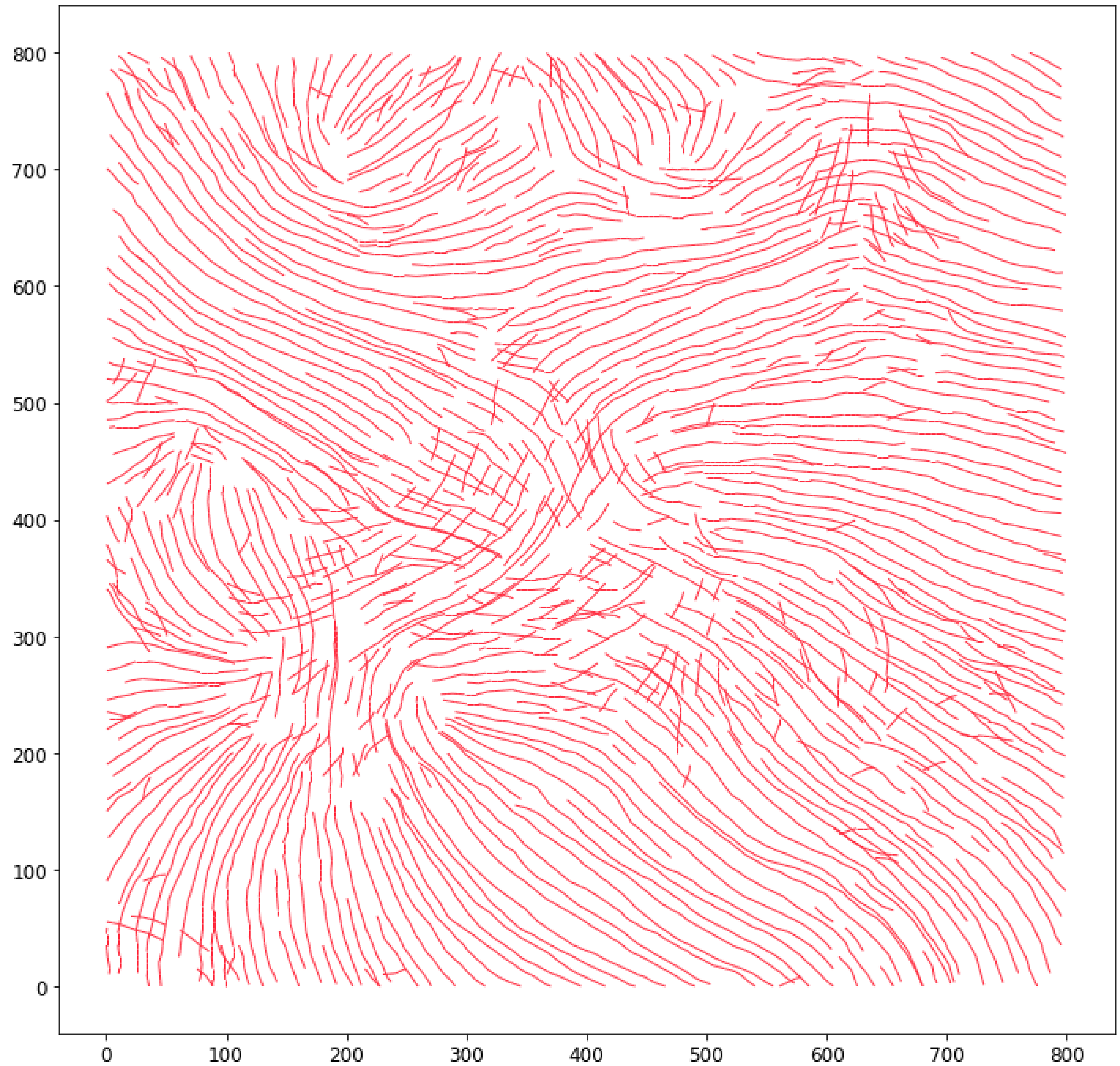
```
65 %            0:00  remaining
70 %            0:00  remaining
75 %            0:00  remaining
80 %            0:00  remaining
85 %            0:00  remaining
90 %            0:00  remaining
95 %            0:00  remaining
Finished in  0:02
100 %            0:00  remaining
Total Time  0:02
Plotting Solid Lines…
```



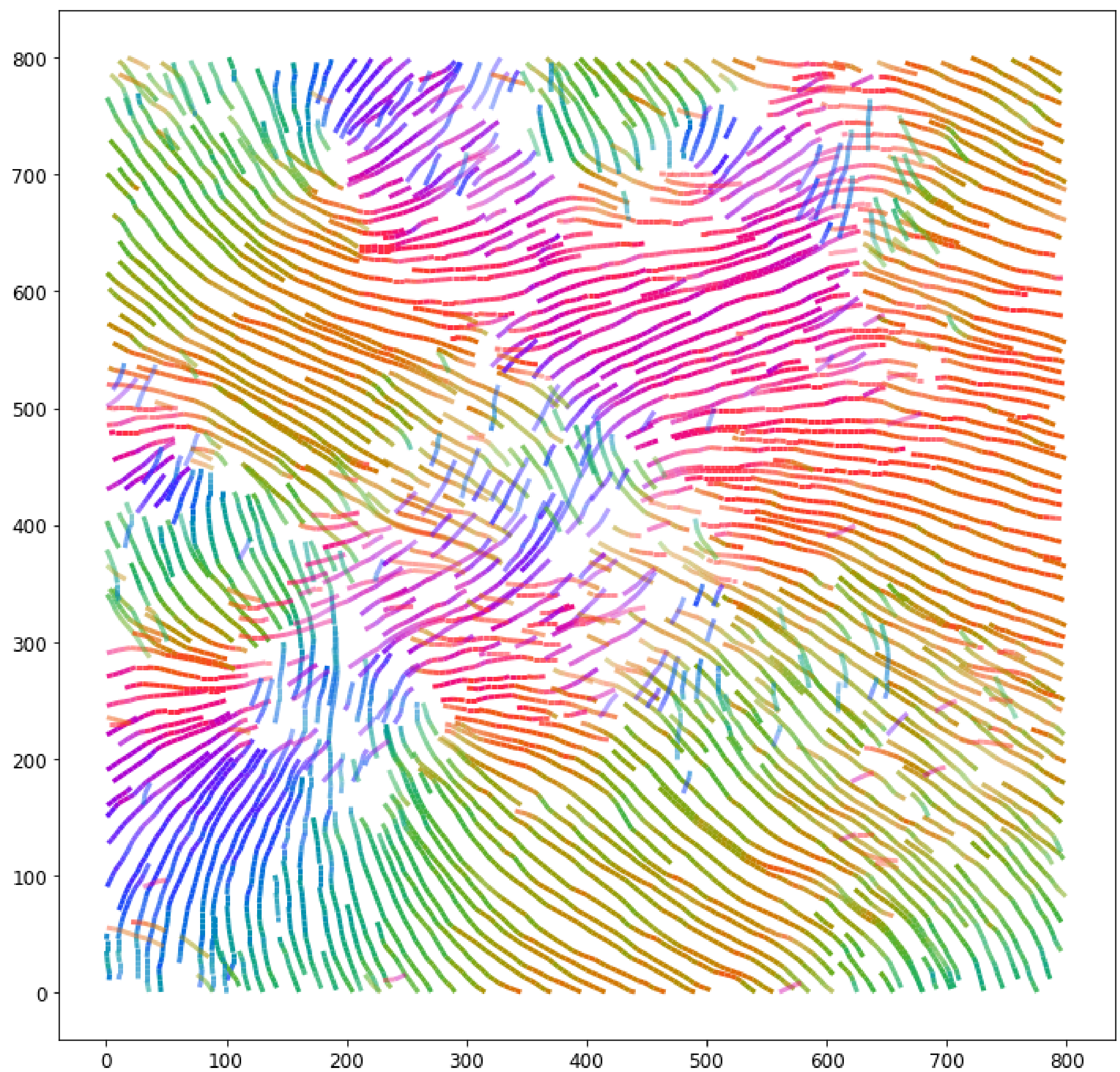
4 Format and Save

[7]:
```
# There are many ways to format the plots.  I suggest keeping settings␣
↪organized in the format below.

format_codes = [0, 1, 2, 3]
formatted_plots = []
contrast = 3
brightness = 1
gamma = 1
for i, format_code in enumerate(format_codes):
    if format_code == 0:
        # Constant color, linewidth, and alpha
        r, g, b = flow.color_by_angle(np.zeros((max_length, n_lines)))
        linewidth = np.ones((max_length, n_lines)) * 1
        alpha = np.ones((max_length, n_lines))
    elif format_code == 1:
        # Color by angle, alpha by intensity
        alpha = flow.scale_values(flow.smooth(intensity_data, 9),␣
↪contrast=contrast, gamma=gamma, brightness=brightness)
        linewidth = np.ones((max_length, n_lines)) * 3
        r, g, b = flow.color_by_angle(angle_data)
    elif format_code == 2:
        # Solid Color, alpha by intensity
        r, g, b = flow.color_by_angle(np.zeros((max_length, n_lines)))
        alpha = flow.scale_values(flow.smooth(intensity_data, 9),␣
↪contrast=contrast, gamma=gamma, brightness=brightness)
        linewidth = np.ones((max_length, n_lines)) * 2
    elif format_code == 3:
        # Faint lines.  Good for overlays.
        # Solid Color, alpha by intensity
        shape = np.ones((max_length, n_lines))
        r = shape * 0.95
        g = shape * 0.95
        b = shape * 0.95
        alpha = flow.scale_values(flow.smooth(intensity_data, 9),␣
↪contrast=contrast, gamma=gamma, brightness=brightness)
        linewidth = np.ones((max_length, n_lines)) * 2


    flow.plot_graded_lines(trimmed_lines, r, g, b, alpha, linewidth)
    formatted_plots.append(plt.gcf())
    plt.show()
```

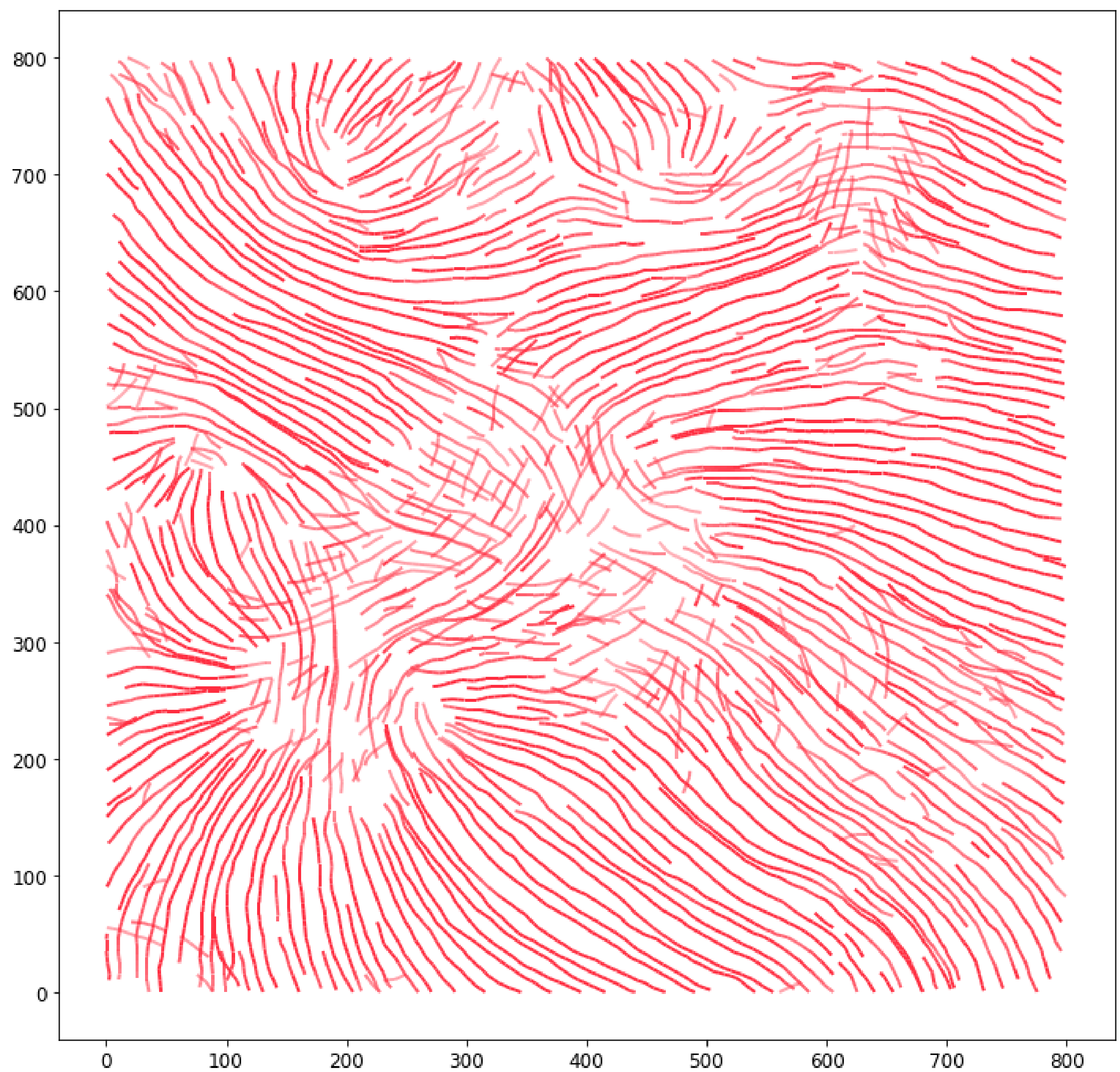
```
Plotting Graded Lines…
```



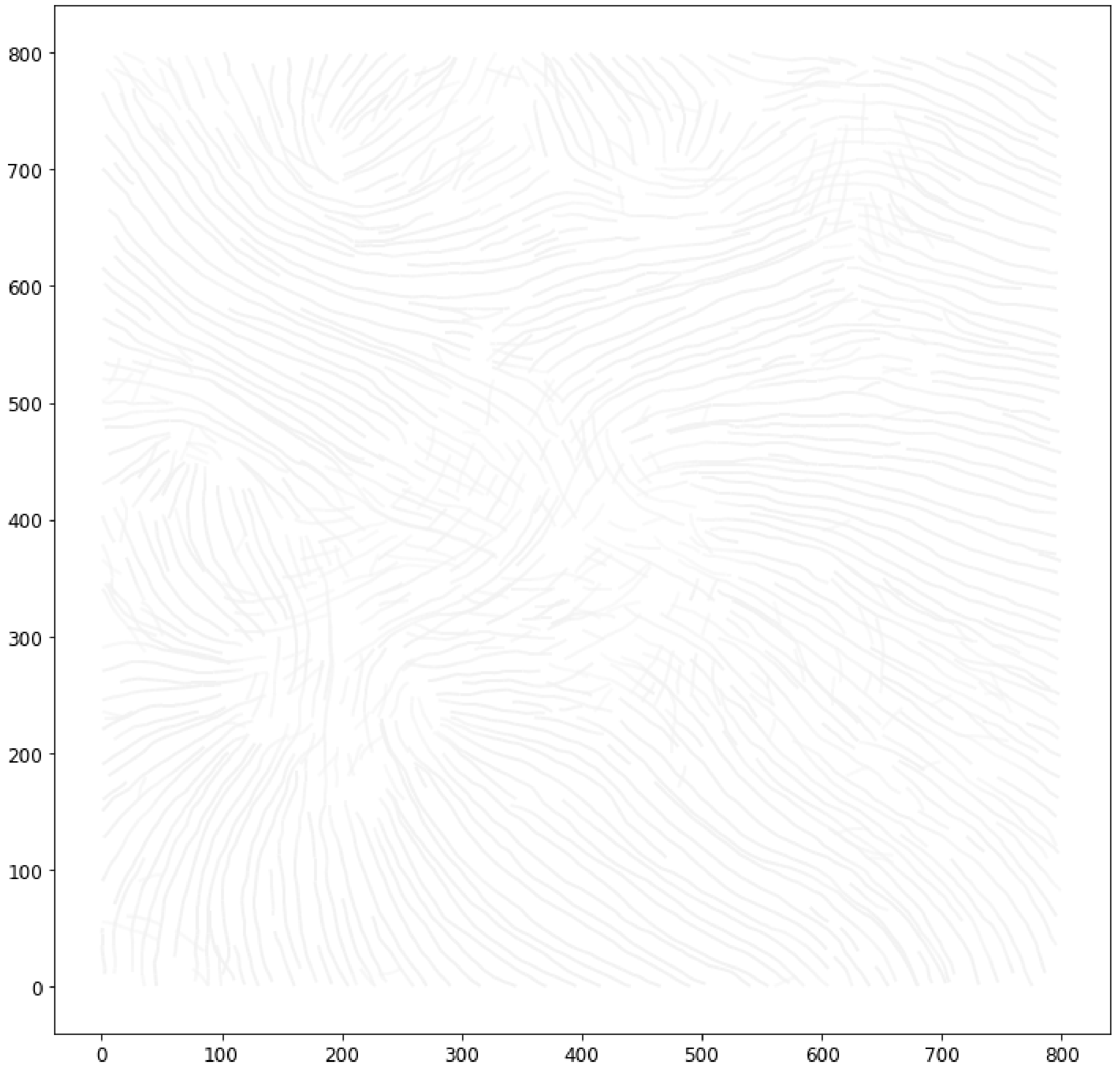
Plotting Graded Lines…



Plotting Graded Lines…



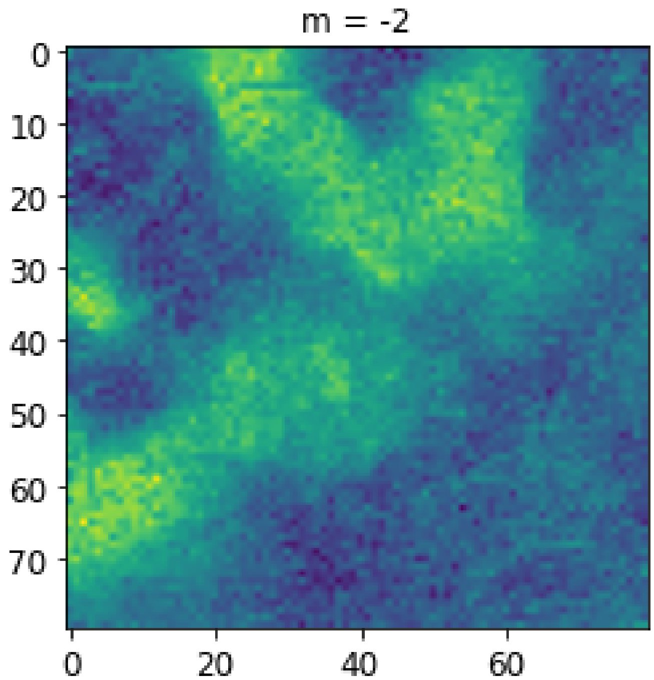

```
Plotting Graded Lines…
```

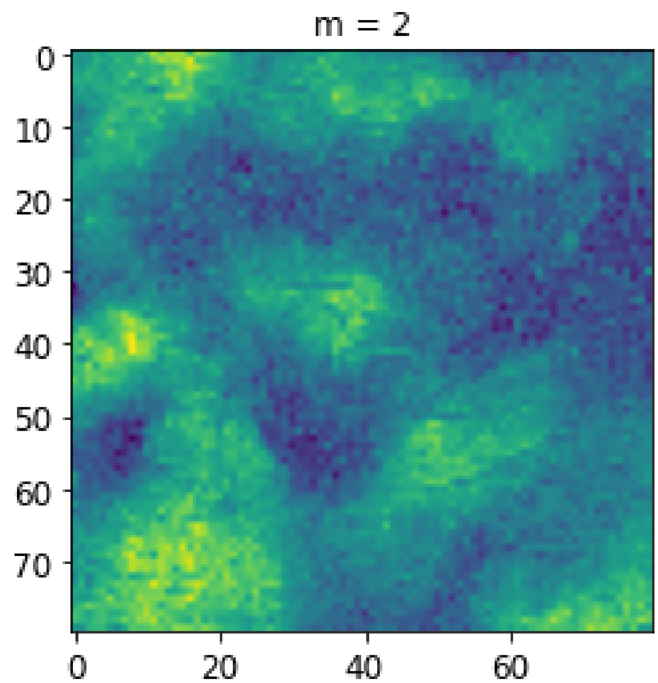


[8]:
```
base_filename = "../figures/sample_3_"

# Save Images
images_to_save = [
    ('propagated_image', propagated_image),
    ('trimmed_image', trimmed_image),
    ('flow_solid', formatted_plots[1]),
    ('flow_color', formatted_plots[2])
]
for name, image in images_to_save:
    image.savefig(base_filename + name + '.png')

formatted_plots[3].savefig(base_filename + 'flow_overlay' + '.png',␣
 ↪transparent=True)
```

5 Spherical Harmonics

[9]:
```
imp.reload(sh)
harmonics = sh.map_to_spherical_harmonics(integrated_intensity, verbose=True)
```



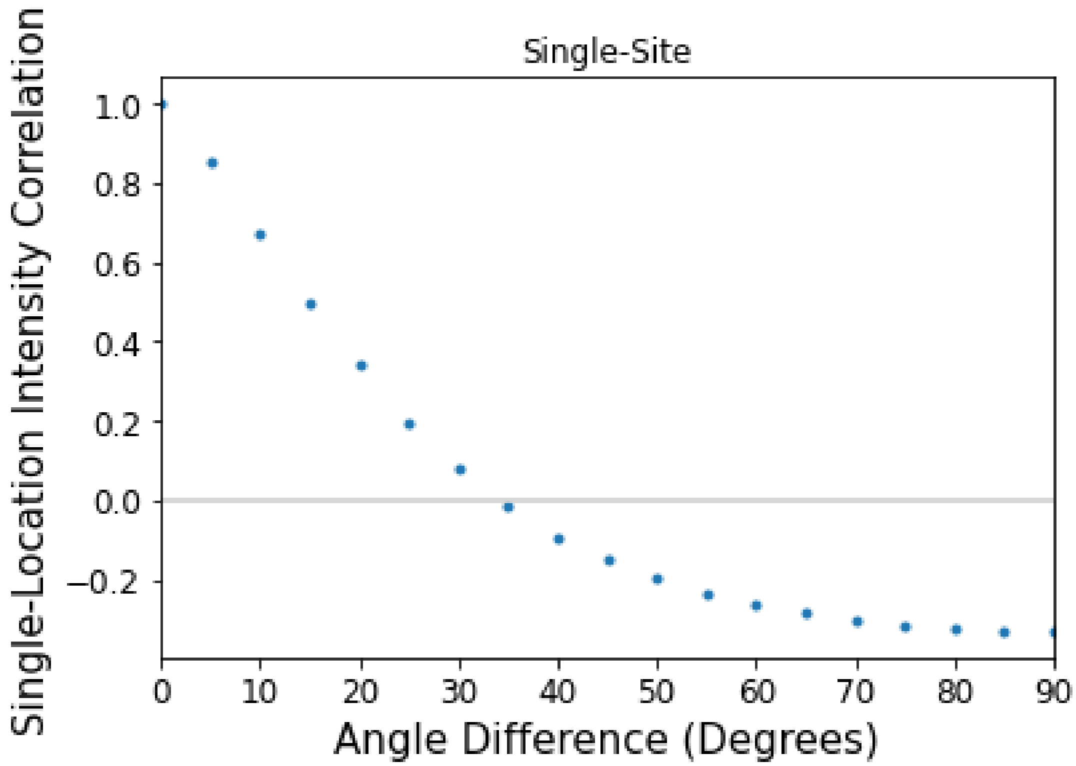


[10]:
```
filename = '../data/spherical_harmonics/sample_3_sh_demo.txt'
np.savetxt(filename, harmonics, fmt='%.5f')
```

6 Autocorrelation

6.1 Single-Location

[11]:
```
imp.reload(ac)
figure_image = ac.single_site_autocorrelation(integrated_intensity)
```



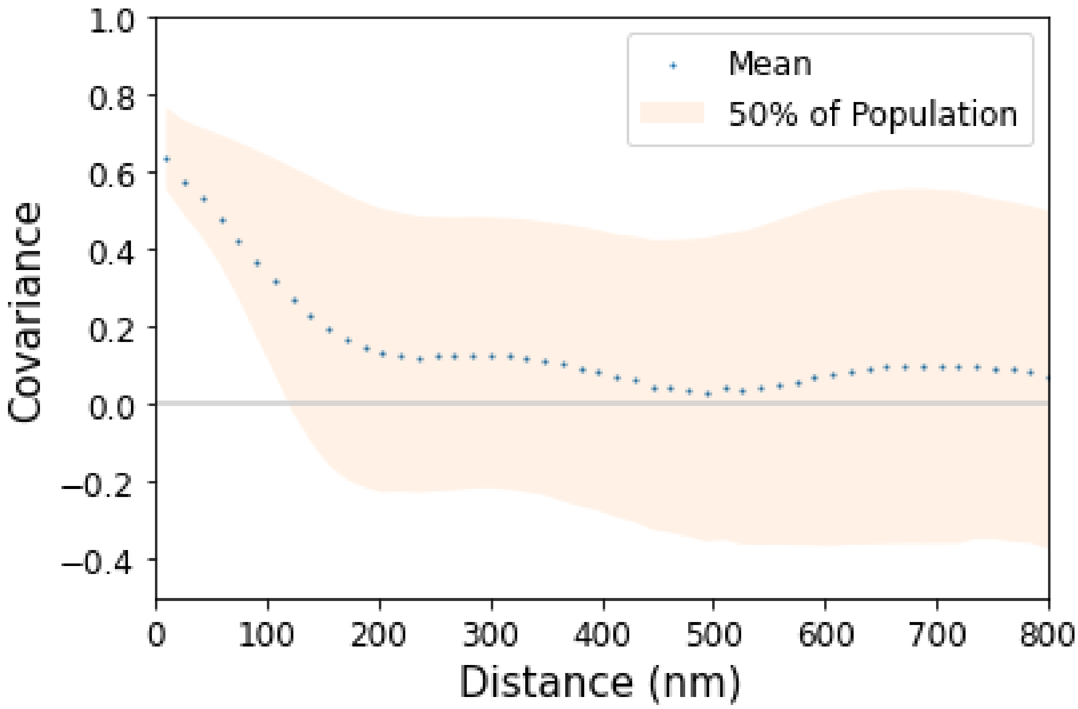
6.2 Resultant Autocorrelation

[12]:
```
imp.reload(ac)
distances = np.linspace(10, 800, 50, endpoint=True)
n_images, n_angles = integrated_intensity.shape
n_grid = int(np.round(n_images**0.5))
intensity_matrix_reshaped = integrated_intensity.reshape(n_grid**2, n_angles)

lc_covariance, stdev, skew, percentiles, scalar_order_parameters = ac.
 ↪resultant_autocorrelation(
    intensity_matrix_reshaped, step_size, distances, verbose=False)
```

```
Setting Up…
Computing Distances…
Computing Angle Differences…
Computing Covariances…
Binning Results…
Dichoric Ratio: 2.3450977795048917
```

[13]:
```
imp.reload(ac)
```

```
lc_covariance_figure = ac.plot_lc_covariance(lc_covariance, percentiles,␣
↪distances)
plt.show()
```



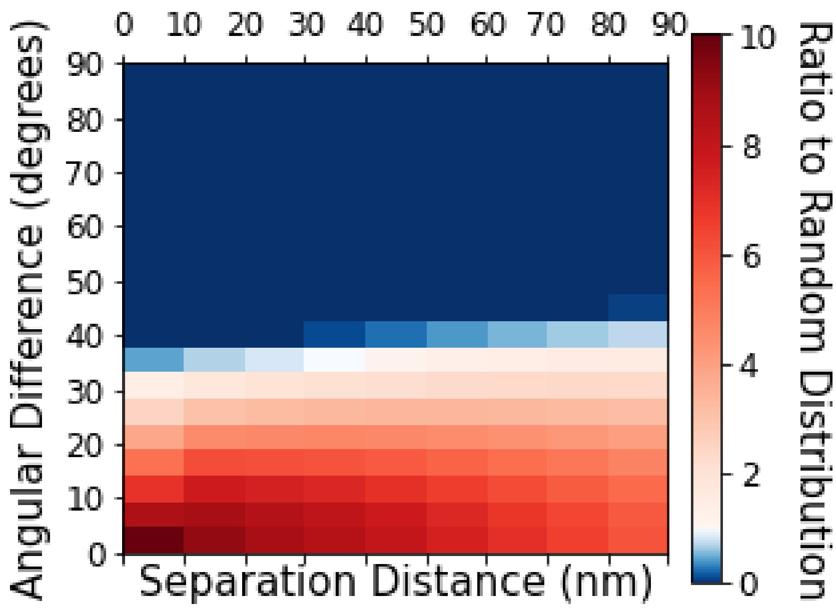
6.3 Autocorrelation with Angular Distributions

[14]:
```
max_grid_distance = 10
grid_distance_step = 1
autocorrelation = ac.distance_angle_autocorrelation(integrated_intensity,␣
↪max_grid_distance, grid_distance_step, n_slices=10)
```

```
Setting Up…
0%
Computing data slice number 0…
9 %           0:14  remaining
Computing data slice number 1…
18 %           0:12  remaining
Computing data slice number 2…
27 %           0:11  remaining
Computing data slice number 3…
36 %           0:09  remaining
Computing data slice number 4…
```

```
45 %            0:08  remaining
Computing data slice number 5…
54 %            0:07  remaining
Computing data slice number 6…
63 %            0:05  remaining
Computing data slice number 7…
72 %            0:04  remaining
Computing data slice number 8…
81 %            0:02  remaining
Computing data slice number 9…
90 %            0:01  remaining
Finished in  0:13
Wrapping up…
(9, 19)
done
```

[15]:
```python
image = ac.plot_distance_angle_autocorrelation(autocorrelation, step_size,␣
 ↪max_grid_distance, grid_distance_step, interpolate=False)
```



[ ]: