THE UNIVERSITY *of* EDINBURGH
School of Physics
and Astronomy

Career Development Summer Scholarship - Project Seagrass

John Whitfield
s2159024@ed.ac.uk

Abstract

This document provides comprehensive documentation for the high throughput nematic ordering software tool developed during my summer project within the Melaugh lab at the University of Edinburgh in collaboration with the charity Project Seagrass. The code was codeveloped with Joseph Knight, who is currently completing a PhD within the Melaugh lab. It includes how to get started, pointers on data acquisition and documentation for the software.

Signature: 

Date: August 25, 2025

Supervisor: Dr. Gavin Melaugh

8 Weeks

Lay Summary

During this project, a piece of software has been developed to analyse microscope images of the protist (single-celled organism) *Labyrinthula zosterae*. It is an organism which lives on seagrass, and was responsible for a large dieback in the 1930s, impacting both local ecosystems and the fishing industry[1]. The software I have developed looks at how the cells in *Labyrinthula* colonies align with one another, a crucial attribute which gives insight to colony (many cells) behaviour. It simplifies microscope images to allow the computer to extract and analyse alignment information. The user can then select an area of the image to see how the alignment changes with distance for that region. Allowing for user input like this both simplifies and increases the speed of the workflow, and serves as a powerful tool. Uncovering alignment helps to understand the physical properties of *Labyrinthula*, working towards the prevention of future seagrass diebacks.

[1] Den Hartog, C. The dieback ofZostera marina in the 1930's in the Waddensea; An eye-witness account by A. van der Werff. Netherlands Journal of Aquatic Ecology 28, 51–54 (1994). https://doi.org/10.1007/BF02334244

Personal Statement

During my 8 weeks in the Melaugh Lab, I vastly improved both my wet lab and microscopy skills, and had the opportunity to produce a high-throughput program for analysing nematic ordering within *Labyrinthula zosterae* colonies. The software tool has been implemented with generality in mind, and is not restricted to the research I conducted this summer. It can be applied to images of any system that exhibits some 2D ordering artefacts.

I now understand better what it means to undertake a PhD within soft matter physics, and it is an option I consider strongly. I have also seen clearly the importance of communication at work, particularly for conveying niche ideas to colleagues working in different areas to myself. This experience has reinforced my passion for research, and I am now left many great decisions to make to shape my future.

Acknowledgements

I want to thank my supervisors, Gavin Melaugh and Joseph Knight, for their great insights and teachings during this 8 week internship. I also extend my gratifications to the University of Edinburgh, who awarded me with the SoPA Summer Career Development Scholarship which funded this project. Finally, I thank Project Seagrass, without whom the project would not have been possible.

Contents

1 Introduction

1.1 Purpose

This documentation acts as an instruction on how to use the high throughput nematic ordering tool designed for use on *Labyrinthula* colonies. It is aimed for those who are working on microscopic biological systems, taking images using optical microscopes, and wish to analyse nematic ordering. That being said, it should work for any images which are composed of some smaller agents which have nematic ordering features which you wish to analyse.

This document will take you through installing the software, setting up a virtual environment, installing the necessary packages, how to preprocess your images for the best results, how to use the software, and the documentation for said software. If there are any features you feel you would like further clarification on, do not hesitate to get in contact with me via the email on the front page of this document.

1.2 Project Overview

This project is a python based image analysis tool, designed to extract nematic ordering information from microscope images of cells and was developed for analysing the dense morphologies sometimes seen in *Labyrinthula zosterae*. The software allows for a user to input an image, and split it into a number of square regions. It will then extract angular information for each of these regions, and calculate from the Q-tensor the nematic ordering parameter, S, and its director. It will overlay the director scaled by the nematic ordering parameter overtop of each of the image regions, producing a 'nematic ordering map'.

The user can then click on any of these cells to calculate a nematic ordering correlation function. This analyses how the nematic ordering varies against distance within that image cell. The software will plot this graph next to the image of the region that has been clicked.

Key Features:

- Image Processing: Skeletonisation, Masking
- Data Extraction: Image moments, Q-Tensor
- Results: Nematic Ordering, Director, Correlation Function

2 Getting Started

Although the code has been developed using a Mac, the below will include instructions of how to set up the software for both MacOS and Windows.

2.1 Creating a virtual environment

To operate on any code, it is good practice to create a virtual environment. This compartmentalises your computer, and allows for better version control on the packages which you need to use. To create and activate (enter into) a virtual environment, open terminal (MacOS) or command prompt (windows) and enter the following commands:

```
# Create a new conda environment
conda create --name [name you want to give your environment]

# Activate environment (Same for Mac/Linux and Windows)
conda activate [name you want to give your environment]
```

Once your environment has been set up and activated, ensure that pip is updated by entering the following command:

```
python -m pip install --upgrade pip
```

Now you are ready to install the required packages to use this software. To complete this, you can copy and paste the following code into your terminal/command prompt window:

Core packages (some come with Python):

```
# Standard library packages (no installation needed)
# os, math, tempfile are part of Python standard library

# Install remaining packages - numpy, matplotlib, scipy
conda install numpy matplotlib scipy
```

Packages for working with images:

```
# For PIL (usually installed as Pillow)
conda install pillow

# For scikit-image
conda install scikit-image

# For OpenCV (cv2)
conda install -c conda-forge opencv
```

Now that your virtual environment and the correct packages are installed, you are ready to download the software onto your system. To do this enter the following command into the same terminal/command prompt window you have been using:

```
# Navigate to Desktop folder (put software files here for easy access)
cd Desktop

# Install software
git clone https://github.com/Tiggger/LabyrinthulaProject
```

You should now have all required packages and a folder on your desktop titled **LabyrinthulaProject**, with an array of files inside, including masterClass.py and imageProcessingTool.py.

3 Recommendations for Image Acquisition

If you are acquiring your own images to input into this software, here are some things to bare in mind when doing so. Firstly, uniform brightness should be prioritised. This is as when you are preprocessing your image, it will suffer significant quality losses when thresholding if illumination is not uniform. This is explained by the fact that thresholding looks at the pixel intensity, $I(x, y)$, of all the pixels that make up the image, and removes all below a certain value. It binarises the image, thus if important features have a range of pixel intensities, when thresholding some of these may be removed.

Furthermore, a bright field image is best. This is for similar reasons as for requiring uniform brightness. Although single cells may be clearer while working in phase contrast, you often get sharp colour changes between phase changes in the image. Due to how thresholding works, this leads to a great loss in detail, and sometimes creates a negative of the cells themselves.

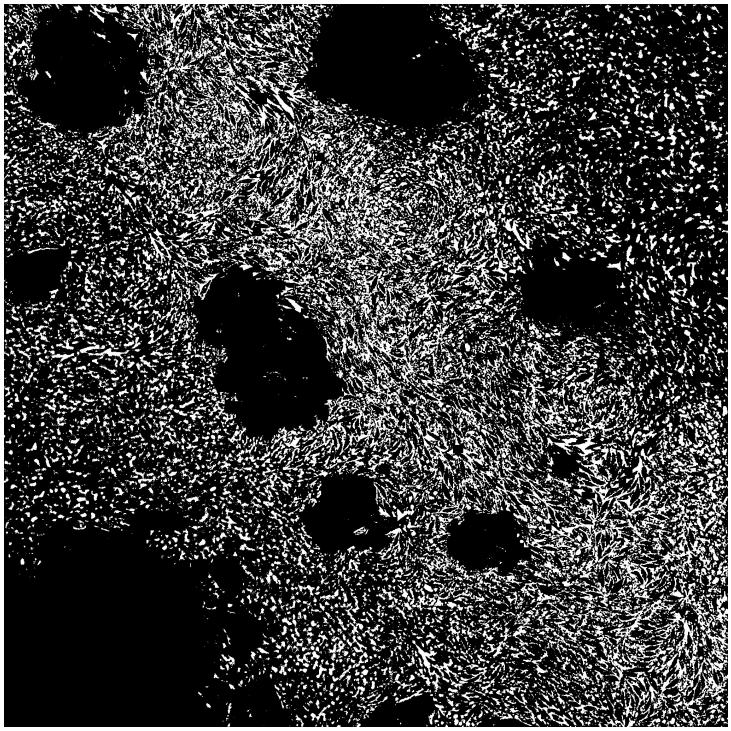

Figure 1: A processed phase contrast image of *Labyrinthula zosterae* taken on an optical microscope using the 20x objective. A great loss of some of the cells has occurred, and you can only see some negatives of other cells in-between these. Although the software will give results, one should questions their reliability.

Different microscopes have different cameras attached to them. Using one with the highest resolution possible is beneficial, as it allows for greater detail to be captured. When looking at *Labyrinthula zosterae* in the dense phase, being able to resolve between the singular cells is a priority. Using a lower resolution camera is not a bad thing, but if possible this should be maximised.

The exposure time of the camera can be altered in micromanager, and the default is 5 ms. If the luminance histogram has been optimised (following Koehler[2] illumination and changing the

[2]For the optical microscopes found on the second floor of the JCMB, this tutorial is very useful.

aperture/brightness), you can increase exposure time and reduce the brightness of the LED/decrease the aperture in succession to ensure the luminance histogram remains in a good position, while increasing the ability to see cells. It was found for *Labyrinthula* colonies that an exposure of 15 ms was ideal.

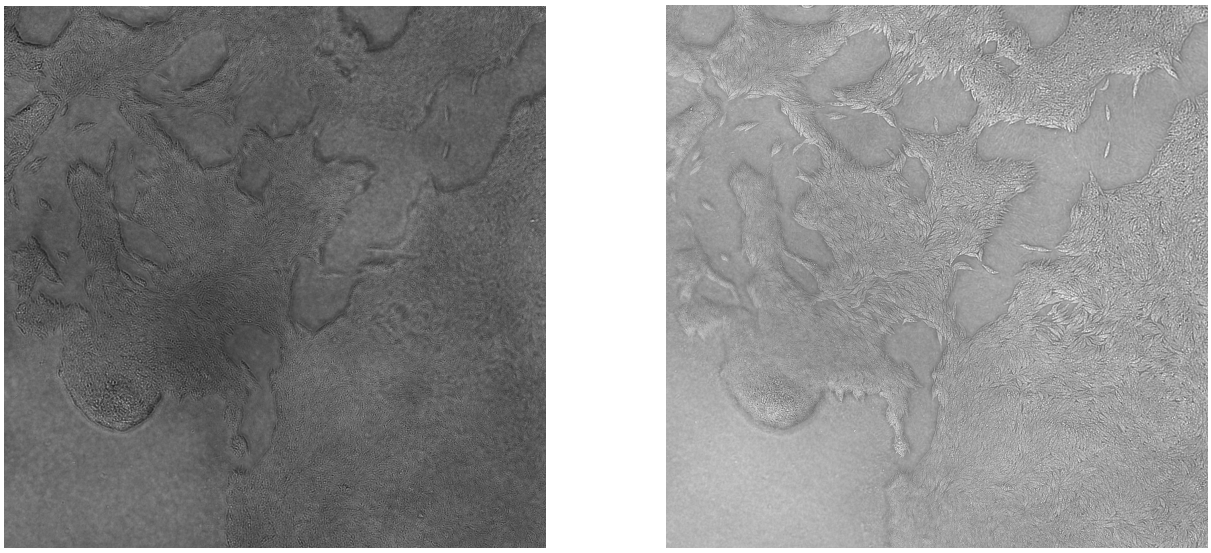

Figure 2: Exposure of 5 ms (left) provides the image with more contrast, but increasing the exposure time to 15 ms increases the cell resolution greatly.

On Nikon 1, a magnification of 40x can be achieved. This provides very clear images of the cells within the dense network morphology, but also amplifies the intensity of the subcellular structures which are captured by the microscope. So although the cell resolution is better, it causes the cells to be a mixture of intensities, and thus when thresholding images (detailed in section 4), crucial detail is lost.

Finally, if possible ensure that most of the field of view is in the same focal plane. This is not always possible, however, it is often possible to search across the sample for a region which is in better focus overall than others. This allows for a more meaningful result in the nematic ordering map, as if you cannot resolve the cells with you eye, the software will also not be able to do so and any results for that region of the image lose their meaning.

A concise list of priorities when acquiring images would be:

- Uniform brightness
- Brightfield Image
- High Resolution Camera
- Increasing exposure time
- Maximising % of image in focus

4 Required Image Preprocessing

Before images can be passed into the software, they must first be preprocessed in Fiji. This process has not been automated, as human judgement is far better than presetting the cleaning parameters. With *Labyrinthula* colonies, resolving the 'cells' at the single cell resolution is achieved far better with a bright field objective. Although it may be easier to see the cells in phase contrast, bright field is far better for the image processing that is involved with the following processes.

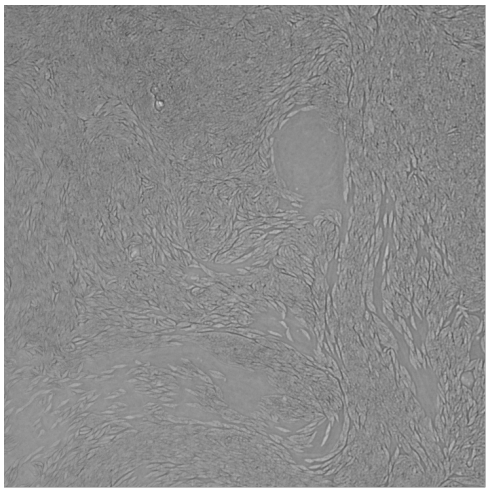

Figure 3: A bright field image of *Labyrinthula zosterae* taken on an optical microscope using the 20x objective.

Beginning from a raw image, the background can be removed using: **Process** > **Subtract Background**, from which a suitable rolling ball radius can be selected. Typically, a smaller radius than the default 50 pixels works better. For *Labyrinthula*, ∼10 pixels is good. Make use of the preview tool to select a suitable radius.

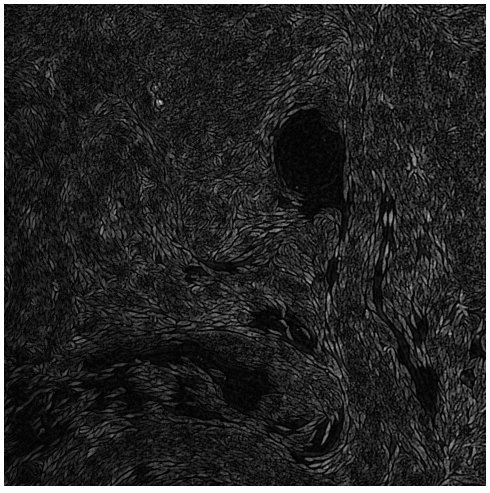

Figure 4: The same image as in Figure 3, but with the background removed using a rolling ball radius of 10 pixels.

If the cells/features you are wishing to analyse look uneven in appearance, as they do in Figure 4, you may wish to apply a Gaussian blur. To do so, head to **Process** > **Filters** > **Gaussian Blur** and select a sigma of 0.5. Again, making use of the preview tool is useful, as the features

in your image may be larger and you can afford to blur more. For *Labyrinthula zosterae* at high magnifications, a sigma of 1 is the highest that can keep onto reasonable cell-level detail.

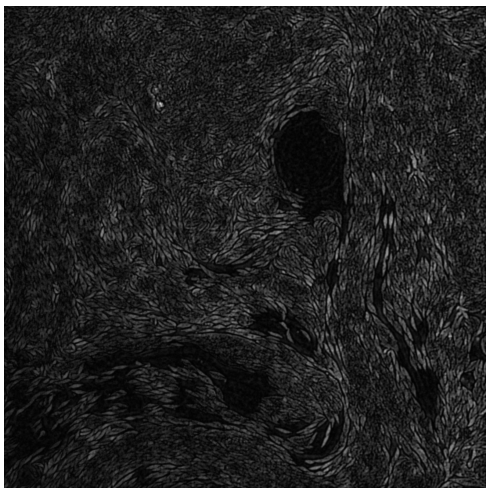

Figure 5: The same image as in Figure 4, but with a sigma = 1 Gaussian blur applied. Please note that the difference is minimal, but does help the end result.

Finally, the image must be thresholded. To apply a threshold to the image, head to **Image** > **Adjust** > **Threshold**. Usually, selecting the auto mode works nicely, however, you may wish to adjust the threshold values using the low and high sliders in order to capture great detail. Once satisfied, you can save the image as a JPEG to a desired location on your computer.

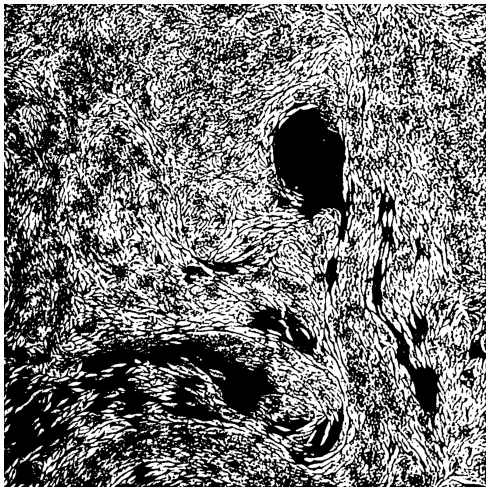

Figure 6: The same image as in Figure 5, but with the auto threshold applied. What is cell and what is not cell is now much clearer for the software to work with. Note that there is still quite a lot of noise, but the image has been far improved.

5 Tool Pipeline Flowchart

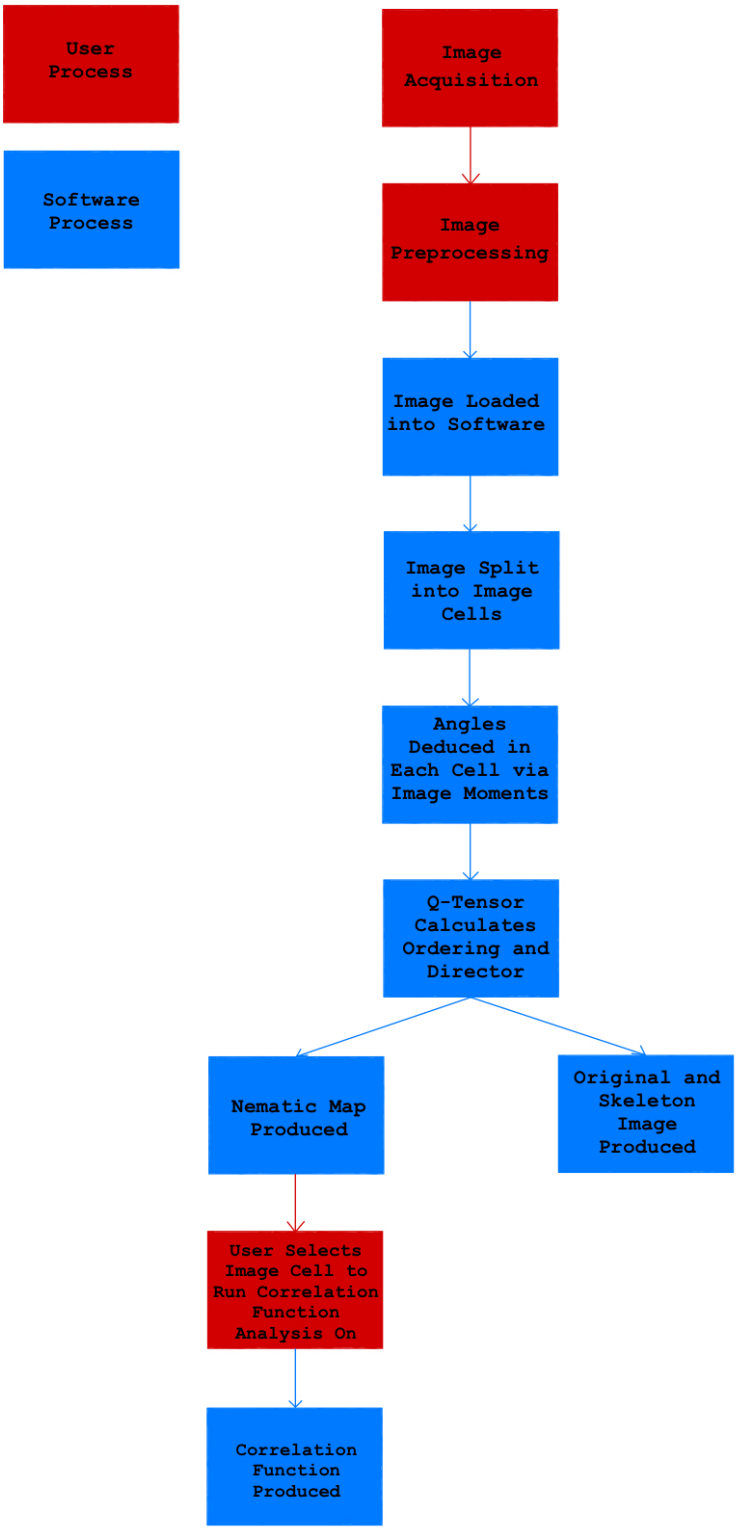

Figure 7: A high-level flowchart detailing the processes that occur in the software.

6 Using The Tool

Once your images have been processed, you can follow **exampleUse.py** to use the tool. You may enter the path to the processed image you wish to analyse, or enter the path of the image within the same git folder the code is in. **Please note you will have to change it from what it is currently set to, as these paths are unique to the machine you are working on**.

The parameters you may wish to change before running are:

- **xsplit**, **ysplit**: The number of divisions in the x and y axes that you wish to split the image into for nematic ordering analysis.
- **kernelSize**: Controls the size of the gaussian kernel which is used within image moment calculations when extracting angular data.

If keeping the image that was provided in the git folder, upon pressing run you will be faced with 2 windows. The first of which is the nematic ordering map.

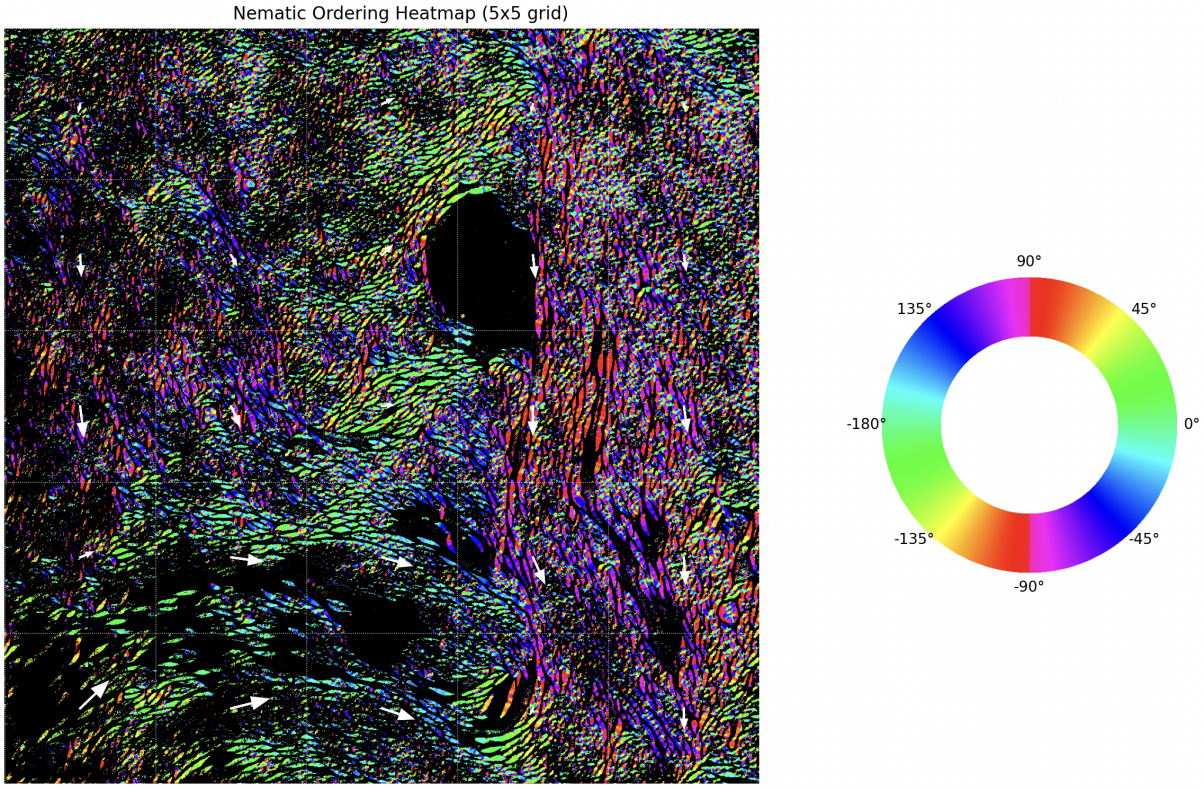

Figure 8: Nematic ordering map generated by the default image in the git folder (left). The image has been divided into a 5x5 grid, as defined by xsplit and ysplit. The image has had the colour of the angles overlayed, and on each grid cell, the director has been plotted in the centre scaled by the nematic ordering parameter, S. The colour wheel (right) is to help identify which portions of the image are oriented at what angle.

If you then click on one of the images in the grid (which has been coined 'image cell'), it will calculate the correlation function for that cell, and output it along with the image in the cell in another window.

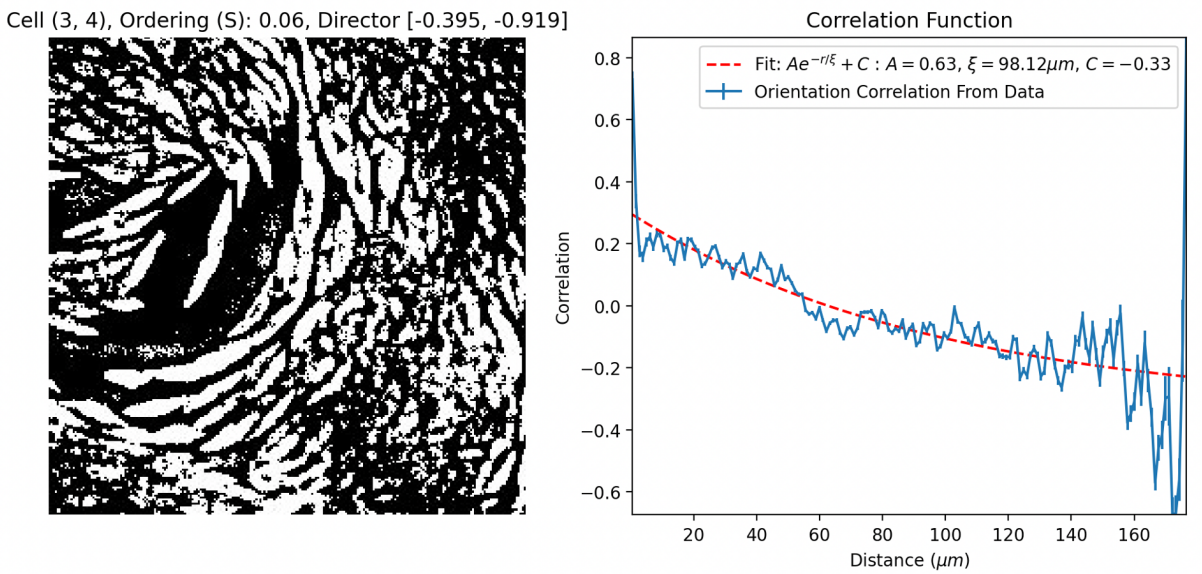

Figure 9: Image in the cell you have clicked, along with its coordinates, S value, and director components (left). The correlation function (right) is also outputted. As long as the correct pixel size has been entered, the x axis will be the distance in microns.

The software will also try to fit an exponential decay to the data. Please note that an exponential decay has been selected as this is appropriate (or hypothesised to be) for *Labyrinthula zosterae*, and you may need to alter what type of fit you want to apply. Paying attention to the long length scales of the right hand side of Figure 9, the data is noisy. The reason for which is that the number of pairs at long length scales is far less than for short length scales. You can continue analysing other cells in the image, or to finish, can close all the windows. From here, you can load other images in etc.

The second window is the original image passed into the software, along with its skeleton. The reason for which is clearer if you read section 8.1.

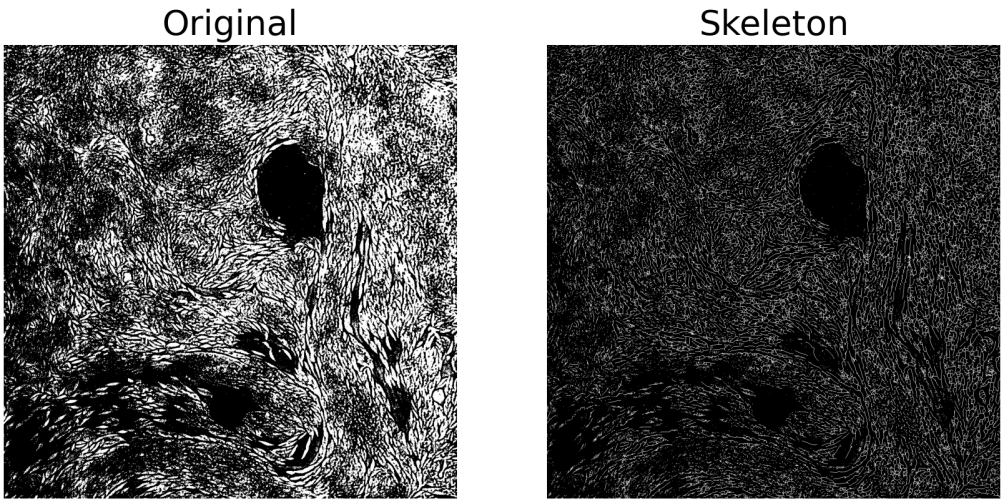

Figure 10: Original image passed into the software (left) and the skeletonised version of it (right). Useful for visualising how well skeletonisation is working.

7 Module Documentation

7.1 masterClass.py

7.1.1 Purpose

The `ImageAnalysis` class provides comprehensive tools for processing input images and extracting angular information it. It includes skeleton processing and orientation analysis. From this it can calculate a correlation function, which is how correlation changes with distance from the centre of the image.

7.1.2 `__init__`

```
def __init__(self, imagePath, skeletonImagePath, radius, sl, threshold=128)
```

Initialises the ImageAnalysis object with necessary parameters and performs initial image processing.

Parameters:

- `imagePath`: Path to the image for analysis
- `skeletonImagePath`: Path to skeletonised version (can be `None` for auto-generation)
- `radius`: Radius for endpoint highlighting (pixels)
- `sl`: Size of Gaussian kernel for angle identification
- `threshold`: Binarisation threshold (default: 128)

Initial Processing:

- Loads and converts image to greyscale
- Processes or generates skeleton image if not provided
- Computes binary image
- Calculates orientation parameters and angle maps
- Generates colour wheel for visualisation

7.1.3 `processSkeleton`

```
def processSkeleton(self)
```

Processes skeletonised images for sparse network analysis and displays results. This is not used for the purposes of what the program achieves, and is a feature that has been left in for adjacent use.

Operations:

- Radially dilates endpoints
- Combines dilated endpoints with original binary skeleton
- Displays four-panel figure showing processing steps

7.1.4 getOrientationImages

```
def getOrientationImages(self, filename)
```

Displays essential pipeline images and saves masked orientation image.

Parameters:

- `filename`: Name for saving masked image

Output:

- Five-panel figure showing original, processed, orientation map, masked orientation, and colour wheel
- Saves masked image to file

7.1.5 calcualteOrientationCorrelation

```
def calcualteOrientationCorrelation(self, coarsening=0)
```

Computes orientation correlation between skeleton points. Achieves this by looking at the angles of points at different distances, and taking a dot product.

Parameters:

- `coarsening`: Controls calculation granularity (higher values reduce computation)

Returns:

- `distance_list`: Pairwise distances between points
- `correlation_list`: Corresponding orientation correlations

7.1.6 binIt

```
def binIt(self, bin_size)
```

Bins the distance-correlation data produced by `calculateOrientationCorrelation` for statistical analysis.

Parameters:

- `bin_size`: Width of distance bins

Returns:

- `bin_centres`: Midpoints of distance bins
- `correlation_avg`: Mean correlation per bin
- `counts`: Number of points per bin
- `std_err`: Standard error per bin

7.1.7 calculateCorrelationAvgNematic

```
def calculateCorrelationAvgNematic(self, correlationAvg)
```

Transforms correlation values to nematic scale ($-0.5 \leq S \leq 1$)

Parameters:

- `correlationAvg`: Raw correlation values

Returns:

- Nematic-scaled correlation values

7.1.8 `plotOrientationCorrelation`

```
def plotOrientationCorrelation(self, bin_centers, correlation_avg_nematic,
    std_err, point_size, xlim, ylim, title, pixelSize)
```

Plots orientation correlation with error bars.

Parameters:

- `bin_centers`: X-axis values
- `correlation_avg_nematic`: Y-axis values
- `std_err`: Error bar values
- `point_size`: Marker size
- `xlim/ylim`: Axis limits
- `title`: Plot title
- `pixelSize`: Distance that 1 pixel represents

7.1.9 `produceCorrelationGraph`

```
def produceCorrelationGraph(self, coarsening, title, pixelSize, bin_size=2,
    plotting=True)
```

Complete pipeline for correlation analysis and visualisation.

Parameters:

- `coarsening`: Calculation granularity
- `title`: Graph title
- `pixelSize`: Distance that 1 pixel represents
- `bin_size`: Bin width (default: 2)
- `plotting`: Toggle visualisation (default: True)

Returns:

- Binned data and statistics

7.1.10 `produceBinCentreGraphs`

```
def produceBinCentreGraphs(self, coarsening, bin_size=2)
```

Generates dual plots of correlation and point counts.

Parameters:

- `coarsening`: Calculation granularity
- `bin_size`: Bin width (default: 2)

7.1.11 Example Usage

```
# Sample code showing how to use this module
import masterClass as mc

image_dir='path/to/image'
skeleton_dir=None
dilatedEndPointRadius=4
kernelSize=4

analysis=mc.ImageAnalysis(image_dir, skeleton_dir, dilatedEndPointRadius,
    kernelSize)

coarsening=10000
title='Correlation Graph'
pixelSize=0.55 #microns

analysis.produceCorrelationGraph(coarsening, title, pixelSize)
```

Listing 1: Instantiating an object and producing a correlation plot. Please note that running this on an image with large dimensions ($\geq$ 1200 pixels) will take a long time for this level of coarsening.

7.2 imageProcessingTool.py

7.2.1 `splitIntoCells`

```
def splitIntoCells(img, xsplit, ysplit)
```

Divides an image into a grid of smaller cells.

Parameters:

- `img`: Input image (greyscale or colour)
- `xsplit`: Number of horizontal divisions
- `ysplit`: Number of vertical divisions

Returns:

- Nested list containing image cells (row-major order)

7.2.2 `calculateDensity`

```
def calculateDensity(cells, binaryImage=False)
```

Calculates cell density from segmented or binary images. Weka should be used for segmentation. Please note that if using the binary image for density, the resulting value should be questioned (found to be unreliable).

Parameters:

- `cells`: List of image cells from `splitIntoCells`
- `binaryImage`: Boolean flag for binary input (default: False)

Returns:

- 2D list of density values (0-1) for each cell

7.2.3 drawCellBoundariesNumpy

```
def drawCellBoundariesNumpy(image, xsplit, ysplit, lineValue=255, thickness=1)
```

Draws grid lines on an image to visualise cell boundaries.

Parameters:

- `image`: Input image
- `xsplit/ysplit`: Grid dimensions
- `lineValue`: Pixel value for lines (default: 255)
- `thickness`: Line thickness in pixels (default: 1)

7.2.4 create_density_heatmap

```
def create_density_heatmap(image, cells, densities, cmap='viridis', alpha=0.5)
```

Generates a static density heatmap overlay.

Parameters:

- `image`: Base image
- `cells`: Cell data structure, created from `splitIntoCells`
- `densities`: Calculated density values
- `cmap`: Colour map (default: 'viridis')
- `alpha`: Transparency (default: 0.5)

Returns:

- Matplotlib figure and axis objects

7.2.5 create_interactive_heatmap

```
def create_interactive_heatmap(image, cells, densities, kernelSize, pixelSize,
    threshold=128, cmap='viridis', alpha=0.5)
```

Interactive density heatmap with click-to-analyse functionality. If a cell is clicked on, the correlation function from `masterClass.py` is ran on that cell.

Parameters:

- `image`: Base image
- `cells`: Cell data structure, created from `splitIntoCells`
- `densities`: Calculated density values
- `kernelSize`: Gaussian kernel size for orientation analysis
- `pixelSize`: Distance that 1 pixel represents
- `threshold`: Binarisation threshold (default: 128)

Returns:

- Matplotlib figure and axis objects

7.2.6 create_interactive_vector_analysis

```
def create_interactive_vector_analysis(image, cells, densities, cmap='viridis',
    alpha=0.5)
```

Interactive density heatmap with the ability to draw a line vector between two points, calculate which image cells this vector crosses, and to calculate the correlation function from `masterClass.py` for all of these cells.

Parameters:

- `image`: Base image
- `cells`: Cell data structure, created from `splitIntoCells`
- `densities`: Calculated density values

Returns:

- Matplotlib figure and axis objects

7.2.7 calculateQTensor

```
def calculateQTensor(cells, kernelSize, threshold=128, batch_size=1000,
    intensityThreshold=0.1)
```

Calculates nematic order parameters using Q-tensor analysis. Please read section 8.1 for more details on the q-tensor.

Parameters:

- `cells`: Cell data structure, created from `splitIntoCells`
- `kernelSize`: Gaussian kernel size for orientation analysis
- `threshold`: Binarisation threshold (default: 128)
- `batch_size`: Processing batch size (default: 1000)
- `intensityThreshold`: Minimum image cell intensity threshold (default: 0.1). Helps to omit calculation of image cell with almost no features in it (physically meaningless)

Returns:

- List containing order parameters and directors for each cell

7.2.8 exponential_decay

```
def exponential_decay(r, A, xi, C)
```

Exponential decay function for correlation fitting.

7.2.9 create_nematicOrderingTensor_heatmap_interactive

```
def create_nematicOrderingTensor_heatmap_interactive(image, cells, orderingInfo
    , kernelSize, pixelSize, colourWheel, threshold=128, coarsening=10000,
    masked_image=None, arrow_scale=0.3, arrows=True, cmap='viridis')
```

Interactive nematic ordering map, where each image cell has the director of the Q-tensor plotted on it, scaled by the nematic odering parameter, also calculated from the q tensor. This function has the ability to calculate the correlation function (from `masterClass.py`) of an image cell.

Parameters:

- `image`: Base image
- `cells`: Cell data structure, created from `splitIntoCells`
- `orderingInfo`: Precomputed ordering data
- `kernelSize`: Gaussian kernel size for orientation analysis
- `pixelSize`: Distance that 1 pixel represents
- `colourWheel`: Reference colour wheel image
- `threshold`: Binarisation threshold (default: 128)
- `coarsening`: Calculation granularity (default: 10000)
- `masked_Image`: Reference to masked image (default: None)
- `arrow_scale`: Director arrow scaling (default: 0.3)
- `arrows`: Toggle director visualisation (default: True)
- `cmap`: Colour map to use (default: viridis)

Returns:

- Matplotlib figure and axis objects

7.3 Synthetic Data Generation

7.3.1 `generate_nematic`

```
def generate_nematic(L, d, N, mode='random', domain_size=None,
    correlation_length=5)
```

Generates synthetic data for testing the nematic ordering functions. You can select from 'random' (randomly oriented rods), 'smooth' (rods whose direction changes over a smooth gradient with distance) and 'domain' (rods who are perfectly aligned in domains). Note that if you wish to test the nematic ordering functions using synthetic data, you need to change the `intensityThreshold` parameter when using `calculateQTensor`.

Parameters:

- `L`: Image size (pixels)
- `d`: Rod length (pixels)
- `N`: Number of rods per dimension
- `mode`: Generation mode ('random', 'smooth', or 'domain')
- `domain_size`: Domain size for 'domain' mode
- `correlation_length`: Smoothing length scale (default: 5)

Returns:

- Generated image and orientation field of the rods

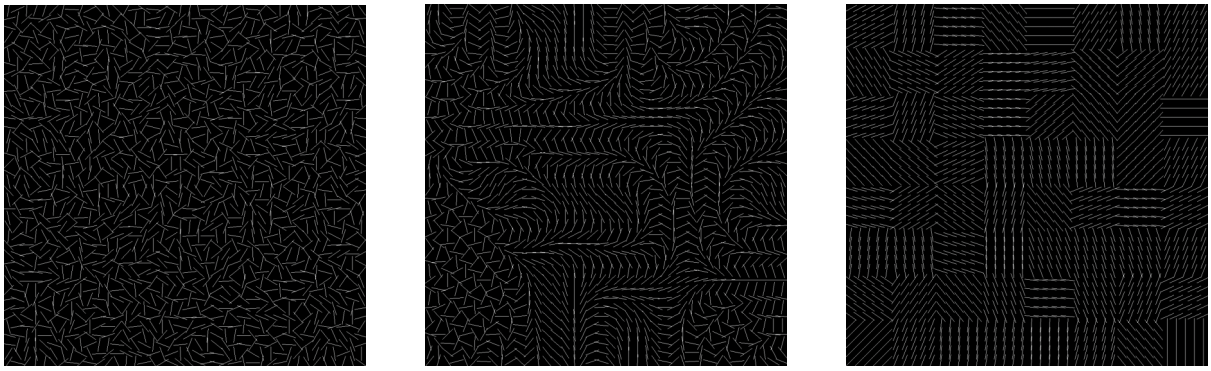

Figure 11: Different synthetic image types. Left: Randomly oriented rods. Middle: Rods whose angle changes smoothly over some domain length. Right: Rods whose angle changes exactly with the domain length.

8 Explaining Calculations

8.1 Image Moments Extracting Angles from Images

8.1.1 Image Moments

Image moments[3] are weighted averages of pixel intensities that provide important geometric properties about the shape and orientation of objects in an image. For a 2D greyscale image with intensity function $I(x, y)$, the raw image moment M_{ij} of order $(i + j)$ is defined as:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

where x and y are the pixel coordinates, and $I(x, y)$ is the pixel intensity at that location.

8.1.2 Central Moments and Orientation

Central moments are translation-invariant and are calculated with respect to the centroid $(\bar{x}, \bar{y})$ of the object:

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y)$$

where $\bar{x} = M_{10}/M_{00}$ and $\bar{y} = M_{01}/M_{00}$ are the centroid coordinates.

The orientation angle θ of the object can be derived from the second-order central moments (μ_{20}, μ_{02}, and μ_{11}):

$$\theta = \frac{1}{2} \arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right)$$

This angle represents the axis of least moment of inertia, which corresponds to the dominant orientation of the object.

8.1.3 Why we calculate them from a skeletonised image

Skeletonisation is a morphological operation that reduces objects in an image to 1-pixel wide representations while preserving their topology and shape characteristics. This process offers several advantages:

- **Increased Aspect Ratio**: By reducing objects to their skeletal form, we effectively increase their aspect ratio (length to width ratio), making orientation detection more accurate and robust.
- **Noise Reduction**: Skeletonisation helps to eliminate small protrusions and irregularities that would be picked up more so in the raw (preprocessed) image.
- **Computational Efficiency**: Working with skeletal representations reduces the number of pixels involved in moment calculations while maintaining the essential shape information.
- **Improved Orientation Detection**: For elongated structures like cells, the skeleton provides a clearer representation of the main axis, leading to more reliable orientation measurements.

[3] I recommend reading more here, should you want to.

8.1.4 Gaussian Weighting in `OrientationFilter` within `Orientation.py`

`Orientation.py` is a hidden away locally-written module, which defines several important functions used for image moment analysis in this software. A Gaussian kernel is used to iterate over the image, and calculate the aforementioned image moments. A Gaussian is used as the kernel for several important reasons:

- **Centre Weighting**: The Gaussian function weights central pixels more heavily than peripheral ones, which is better suited for identifying orientations, especially with a skeletonised image.
- **Noise Suppression**: The smooth falloff of the Gaussian suppresses noise while preserving important structural features.
- **Scale Control**: The standard deviation σ of the Gaussian allows control over the spatial scale of orientation analysis.

The Gaussian kernel $G(x, y)$ is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

8.1.5 `OrientationFilter` Details

The orientation analysis is implemented through the following key operations:

1. **Gaussian Kernel Construction**: Creates a 2D Gaussian kernel with $\sigma = \texttt{sl}/2$
   - `sl` is the parameter name of the size of the kernel
2. **Convolution Operations**: Computes various weighted moments through convolution:
   - `wt`: Gaussian-weighted intensity
   - `xw`, `yw`: Intensity-weighted coordinates
   - `x2w`, `y2w`, `xyw`: Second moments
3. **Angle Calculation**: Derives the orientation angle ϕ:

$$\phi = \frac{1}{2} \arctan 2\left(\frac{-2(\mu_{11})}{\mu_{20} - \mu_{02}}\right)$$

4. **Nematic Order Parameter**: Computes a measure of orientation strength, with values closer to 1 indicating strong alignment and values near 0 indicating isotropic regions.

8.2 Q Tensor Calculation for Ordering Map

8.2.1 Introduction to the Q-Tensor

The Q-tensor is a symmetric, traceless order parameter that describes nematic ordering in liquid crystals and other anisotropic systems. For 2D orientation fields, it is defined as:

$$Q = \begin{bmatrix} Q_{xx} & Q_{xy} \\ Q_{xy} & Q_{yy} \end{bmatrix} = S \begin{bmatrix} n_x^2 - \frac{1}{2} & n_x n_y \\ n_x n_y & n_y^2 - \frac{1}{2} \end{bmatrix}$$

where:

- S is the scalar order parameter (degree of alignment)
- $\mathbf{n} = (n_x, n_y)$ is the director (average orientation)

8.2.2 Mathematical Formulation

The Q-tensor components are computed from orientation angles θ (which are extracted from image moment analysis) as:

$$Q_{xx} = 2\langle \cos^2 \theta \rangle - 1$$
$$Q_{xy} = 2\langle \cos\theta \sin\theta \rangle$$
$$Q_{yy} = 2\langle \sin^2 \theta \rangle - 1$$

where as usual, $\langle\rangle$ is spatial averaging. The order parameter (S) and director are obtained from the eigenvalue decomposition of Q. S is the largest eigenvalue of the Q-tensor matrix, and the director is the corresponding eigenvector to this eigenvalue.

8.2.3 Implementation Details

The `calculateQTensor` function in `imageProcessingTool.py` processes a grid of cell images through these steps:

1. **Cell Preprocessing**:
   - Filters out dark cells (`intensity` < `threshold`)
   - Saves each cell as a temporary image for analysis
   - Computes orientation field using `ImageAnalysis`
2. **Batch Processing**:
   - Processes angles in batches to manage memory[4]
   - Accumulates tensor components:

$$\texttt{sum\_xx} = \sum \cos^2 \theta$$
$$\texttt{sum\_xy} = \sum \cos\theta \sin\theta$$
$$\texttt{sum\_yy} = \sum \sin^2 \theta$$

[4]Processing would have been completed numpy style, but on my 8GB RAM system, I had memory issues.

3. **Q-Tensor Construction**:
   - Computes averages:

$$\begin{aligned} \texttt{avg\_xx} &= \texttt{sum\_xx}/\texttt{total} \\ \texttt{avg\_xy} &= \texttt{sum\_xy}/\texttt{total} \\ \texttt{avg\_yy} &= \texttt{sum\_yy}/\texttt{total} \end{aligned}$$

   - Builds Q-tensor matrix:

$$Q = \begin{bmatrix} 2 * \texttt{avg\_xx} - 1 & 2 * \texttt{avg\_xy} \\ 2 * \texttt{avg\_xy} & 2 * \texttt{avg\_yy} - 1 \end{bmatrix}$$

4. **Eigenanalysis**:
   - Computes eigenvalues λ_1, λ_2 and eigenvectors
   - Extracts order parameter $S = \max(\lambda_1, \lambda_2)$
   - Determines director from eigenvector of maximal eigenvalue[5]

8.2.4 Parameters and Output

- **Input Parameters**:
  - `cells`: 2D array of cell images
  - `kernelSize`: Size of orientation analysis kernel
  - `threshold`: Intensity cutoff for valid pixels
  - `batch_size`: Controls memory usage during processing
  - `intensityThreshold`: Minimum mean intensity for analysis
- **Output**:
  - Returns 2D list of `[S, director]` pairs matching input grid
  - $S \in [0, 1]$ where:
    - 0 = isotropic disorder
    - 1 = perfect alignment
  - `director` is a normalised vector of average orientation for a given image cell

8.2.5 Physical Interpretation

The Q-tensor analysis reveals:

- **Local alignment strength** through S
- **Preferred orientation** through the director
- **Structural anisotropy** of cellular organization
- **Defect locations** where $S \approx 0$

[5]Please note that when these are being plotted they have been reflected in the y direction, as `matplotlib`'s imshow has the origin in the top left rather than the bottom left.

8.3 Calculation of Correlation Orientation Graph

8.3.1 Orientation Correlation Calculation

The `calculateOrientationCorrelation` method computes the spatial correlation of orientation vectors across the image. This analysis helps quantify how orientation patterns vary with distance in the sample.

8.3.2 Method Overview

The function performs pairwise comparisons between orientation vectors at different positions in the image, calculating both their spatial separation and angular correlation.

8.3.3 Mathematical Formulation

For two points at positions (x_1, y_1) and (x_2, y_2) with orientation angles θ_1 and θ_2, the following quantities are computed:

- **Distance** between points:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Orientation correlation** (squared cosine similarity):

$$C = |\cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)|^2 = |\vec{v}_1 \cdot \vec{v}_2|^2$$

  where $\vec{v}_i = (\cos\theta_i, \sin\theta_i)$ are the unit orientation vectors.

8.3.4 Implementation Details

The algorithm proceeds as follows:

1. **Input Processing**:
   - Extracts pixel coordinates from the skeleton image where orientation is defined
   - Applies coarsening by subsampling points when `coarsening > 0`
2. **Correlation Computation**:
   - For each reference point (potentially subsampled):
     - Converts orientation angle to unit vector
     - Compares with all other points in the image
     - Computes distance and correlation for each pair
   - Special case: auto-correlation (distance = 0) is set to 1
3. **Output**:
   - Returns two lists:
     - `distance_list`: Pairwise distances between points
     - `correlation_list`: Corresponding orientation correlations

8.3.5 Parameters and Optimisation

Key features of the implementation include:

- **Coarsening**: The `coarsening` parameter controls computational intensity by:
  - Skipping points in the outer loop (reference points)
  - Maintaining full resolution for comparison points
  - Effectively sampling every $(n + 1)$-th point where n is the coarsening factor

8.3.6 Physical Interpretation

The correlation function provides insight into:

- **Orientation coherence length**: Distance over which orientations remain correlated
- **Anisotropy domains**: Regions of similar orientation
- **Structural organisation**: Degree of global alignment in the sample

8.4 Statistical Analysis of Correlation data

The `binIt` method performs statistical analysis of orientation correlations by binning data based on spatial distance. This enables systematic study of how orientation correlations vary with separation distance.

8.4.1 Method Overview

The function:

- Groups pairwise correlation measurements into distance bins
- Computes average correlation and error statistics for each bin
- Handles edge cases and empty bins appropriately

8.4.2 Mathematical Operations

For a given bin size Δd, the method:

1. Defines distance bins:

$$B_i = [i\Delta d, (i+1)\Delta d), \quad i = 0, 1, \ldots, \left\lfloor \frac{d_{\max}}{\Delta d} \right\rfloor$$

2. Computes bin centers:

$$d_i = \left(i + \frac{1}{2}\right) \Delta d$$

3. Calculates average correlation for each bin, where $C(d)$ is the correlation for a given distance:

$$\langle C \rangle_i = \frac{1}{N_i} \sum_{d \in B_i} C(d)$$

   where N_i is the number of points in bin B_i.

4. Computes standard deviation:

$$\sigma_i = \sqrt{\frac{1}{N_i - 1} \sum_{d \in B_i} (C(d) - \langle C \rangle_i)^2}$$

5. Calculates standard error of the mean:

$$\text{SEM}_i = \frac{\sigma_i}{\sqrt{N_i}}$$

8.4.3 Implementation Details

The algorithm proceeds through these steps:

1. **Bin Initialisation**:
   - Determines maximum distance in the dataset
   - Creates equally spaced bins covering the full distance range
   - Initialises storage for statistics
2. **Data Binning**:

- Iterates through all distance-correlation pairs
- Assigns each pair to the appropriate distance bin
- Maintains separate lists of raw values for each bin

3. **Statistical Calculations**:
   - Computes mean correlation for each non-empty bin
   - Calculates standard deviation of correlations within each bin
   - Derives standard error of the mean
   - Handles empty bins by setting values to `np.nan`

8.4.4 Parameters and Output

- **Input Parameters**:
  - `bin_size`: Controls spatial resolution of correlation analysis
    - Smaller values give higher resolution but noisier results
    - Larger values provide smoother trends but may obscure features
- **Output**:
  - `bin_centers`: Midpoints of distance bins
  - `correlation_avg`: Mean correlation in each bin
  - `counts`: Number of data points per bin
  - `std_err`: Standard error of the mean correlation

8.4.5 Physical Interpretation

The binned correlation function reveals:

- Characteristic length scales of orientation ordering
- Distance at which correlations decay significantly
- Potential periodic patterns in orientation alignment
- Overall coherence of the nematic order

The standard error estimates provide confidence intervals for interpreting the correlation trends.

9 Troubleshooting

9.1 Common Issues

9.1.1 Incorrect Kernel Size

Selecting a correct value for the `kernelSize` parameter is important for getting a meaningful orientation field of an image. The way that the program calculates the angles is via image moments on the **skeletonised image**.

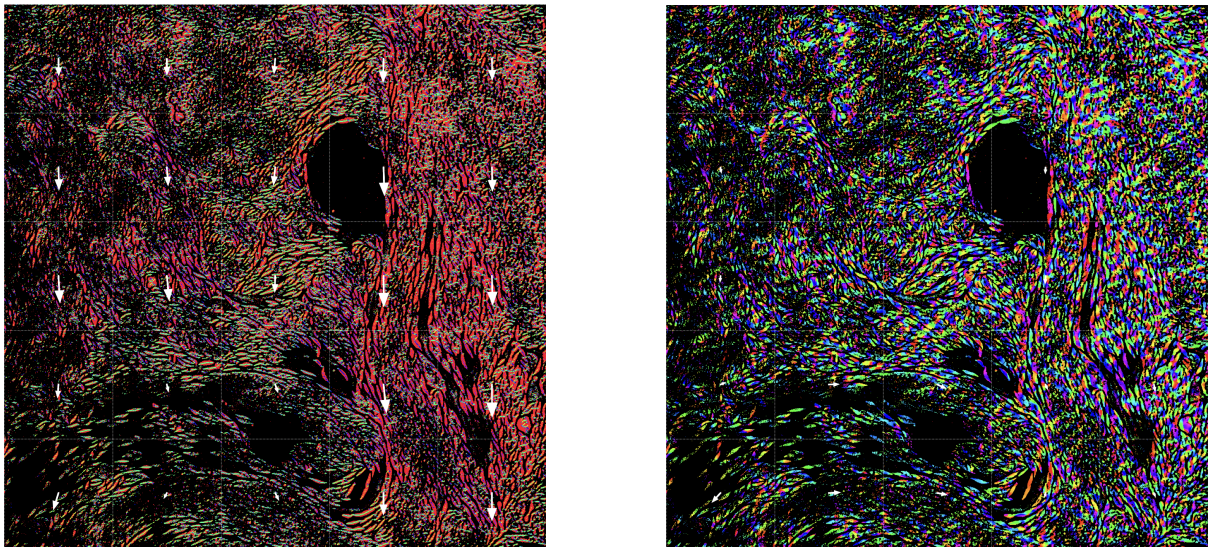

Figure 12: Left: A kernel size (1 pixel) that is too small provides the image with lots of 'null' angles (red), which is due to the kernel size being much smaller than the size of a cell. Right: A kernel size (15 pixels) that is too large, and provides the orientation field with lots of noise. This is due to the kernels overlaying each other, and thus getting 'blended' together when the program iterates over the image. The ill choice of kernel sizes is reflected in both images with incorrect directors.

If the kernel size is too small, you will see a lot of red in your masked image. The reason for this is when the Gaussian kernel iterates over the image, it will capture the skeleton alone, but will not be large enough to also cover the surrounding areas. This will cause for those areas to be assigned the default value ($\pm 90°$), which is coloured red. Please note that if cells are pointing vertically, then red is the correct colour, however, for cells which are evidently pointed in other directions, red shouldn't be seen. When the original image is then masked back over the orientation field, you will be able to clearly see the skeleton

Figure 13: Comparison of the original image, skeletonised image, and the resulting masked image from using the same kernel size (1 pixel) as in the left hand side of Figure 12. Within the masked image, it is clear where the skeleton lays (the coloured strip within the red).

If instead the kernel size is too large, a lot of noise will be seen in the masked image. The reason for which is that the larger Gaussian kernel will cover several skeletonised cells simultaneously, and will also overwrite previously assigned angles. This leads to regions that are cells being a large mix of colours, none of which particularly meaningful for the direction of cells. This is fairly easy to deduce by eye.

Figure 14: Comparison of the original image, skeletonised image, and the resulting masked image from using the same kernel size (15 pixels) as in the right hand side of Figure 12. Within the masked image, there are several colours for one cell, which cannot be the case as the cell clearly points in on direction.

Please note that for a larger kernel size, the program takes longer to run. This is due to for each point of analysis within the image, the program is having to evaluate a larger area as the dimensions of the Gaussian kernel are larger. Typically, for a 20x image, a kernel size of 4 works quite well.