Quantifying Zebrafish Patterns: Code Documentation

Melissa R. McGuirl, Alexandria Volkening, Björn Sandstede
contact: melissa_mcguirl@brown.edu

This documentation corresponds to code for quantifying spots and stripe pattern features using topological data analysis and machine learning. The code is publicly available on Github. Complete datasets from full model simulations are freely available on Figshare. See our paper for further details.

Contents

1 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

1.1 Prerequisites

This program has the following requirements:

- Ripser.py v0.3.2
- Python (2.7 or greater, but we recommend using Python3) with the following libraries:
  - cython
  - ripser
  - matplotlib
  - numpy
- Matlab (R2017 or greater)

1.2 Installation

First, to obtain the code from Github on your local machine:

```
$ cd
$ git clone https://github.com/sandstede-lab/Quantifying
    _Zebrafish_Patterns
$ cd Quantifying_Zebrafish_Patterns
```

To install the prerequisites we recommend using pip installation as follows:

```
$ pip install -r requirements.txt
```

Or, you can manually install the prerequisites:

```
$ pip install Cython
$ pip install Ripser
$ pip install matplotlib
$ pip install numpy
```

Note, if you are using Python3, you may need to use 'pip3' instead of 'pip'.

1.3 Understanding the repository

There are two main folders in the repository: src and data. The data folder contains sample input data for testing purposes. The src folder contains all of the code, separated into 'python' and 'matlab' subfolders. Most of the code in this repository is written in Matlab. We only use Python to apply Ripser, which computes the persistence diagrams of the cell coordinates.

Within the 'src/matlab' folder, the two main scripts are 'quantify_spots.m' and 'quantify_stripes.m,' which output statistics of spots/stripes given cell coordinate data and corresponding persistence diagrams as inputs. There are also two subfolders within the 'src/matlab' folder. First, the 'src/matlab/examples' folder provides four sample Matlab scripts which demonstrate how to run the entire pipeline, starting from pre-saved model

simulation data. Second, 'src/matlab/util' houses the utility scripts for our methods.

1.4 Data samples

Simulation data: MAT files from simulations of the agent-based model of A.V. and B.S. under the default parameter regime are provided as sample data in the 'data/sample_inputs/' folder. Complete datasets from large-scale model simulations are also freely available on figshare.

Distance matrix data: Text files of pair-wise distance matrices between cell coordinates are provided in 'data/sample_dist_mats/'. This folder contains distance matrices for the melanophore (melD_WT_Test_day46.txt), dense xanthophore (xanD_WT_Test_day46.txt), and loose xanthophore (xanL_WT_Test_day46.txt) cell coordinates from the final time point (day 46) of a default wild-type simulation, the melanophore cell coordinates from the final time point (day 76) of a default *shady* simulation (melD_Shady_Test_day76.txt), and the loose iridophore cell coordinates from the final time point (day 56) of default *pfeffer* (iriL_Pfeffer_Test_day56.txt) and *nacre* (iriL_Nacre_Test_day56.txt) simulations.

All distances are computed with periodic boundary conditions in the x-direction (horizontal direction). These distance matrices are used as input to Ripser to compute the corresponding persistence diagrams.

Persistence diagram data: Text files containing the persistence diagram data for each of the sample simulations are available in 'data/sample_barcodes/'. This folder contains the dimension 0 and dimension 1 persistence diagrams corresponding to the melanophores (melD_WT_Test_day46_dim*), dense xanthophores (xanD_WT_Test_day46_dim*), and loose xanthophores (xanL_WT_Test_day46_dim*) from a default wild-type simulation, the dimension 0 and dimension 1 persistence diagrams corresponding to melanophore cell coordinates from a default *shady* simulations (melD_Shady_Test_day76_dim*), and the dimension 0 and dimension 1 persistence diagrams of the loose iridophores from default *nacre* (iriL_Nacre_Test_day56_dim*) and *pfeffer* simulations (iriL_Pfeffer_Test_day56_dim*).

Each text file contains two columns of real numbers. The first column corresponds to the birth radius of each persistence feature in the data, and the second column corresponds to the death radius. Each row is therefore a point in the persistence diagram whose coordinate values are given by the values in the first and second columns. In practice, usually you will only need the dimension 1 persistence diagrams for stripe quantification and the dimension 0 persistence diagrams for spot quantification.

1.5 Example runs

Example scripts are provided to demonstrate how to generate all of the required input data (starting from a model simulation) and how to run the program to quantify wild-type stripes and mutant spots. Go into the 'src/matlab/examples' folder to get started.

```
$ cd src/matlab/examples
```

After installing all of the prerequisites you should now be able to quantify the provided simulation data of wild-type, *shady*, *pfeffer*, and *nacre* patterns. To quantify the wild-type simulations run 'test_WT.m.' Similarly, to quantify the mutant simulations run 'test_shady.m', 'test_pfeffer.m',

and 'test_nacre.m'. You can run these test scripts directly without having to modify any code by simply typing 'test_WT.m' into the Matlab command window, and similarly for 'test_shady.m', 'test_pfeffer.m', and 'test_nacre.m'.

We now explain the code within these test scripts. The pipeline to go from the model simulation outputs to detailed pattern quantification is as follows:

1. Load in the .mat file from your model simulation

```
input_dir =
    '../../../data/sample_inputs/Out_WT_default_1.mat';
load(input_dir)
```

2. Extract cell coordinates for each cell type at the final time point of the simulations and remove cells near the boundary (time_pt varies for wild-type and the mutant fish in these simulations; it is defined separately within each test file),

```
cutoff =  0.1*boundaryY(time_pt);
cells_mel = cellsM(find(cellsM(1:numMel(time_pt),
    2,time_pt)> cutoff &  cellsM(1:numMel(time_pt), 2,time_pt)
    < boundaryY(time_pt) - cutoff), :, time_pt);
                          .
                          .
                          .
```

3. Compute pair-wise distances between cells and save the resulting distance matrices as text files

```
[D_mel, ~, ~] = getPeriodicDistMats(cells_mel,
    boundaryX(time_pt));
savename = 'WT_Test';
path2outputs = '../../../data/sample_dist_mats';
dlmwrite([path2outputs '/melD_' savename '_day'
    num2str(time_pt) '.txt'], D_mel, 'precision',3)
```

4. Plot simulated patterns (optional)

5. Use Python to run Ripser on the cellular distance matrices from step 3. **Note: This step requires you to step out of Matlab and into a terminal window.**

```
$ python3 get_barcodes.py -i PATH_TO_DISTANCE_MATRIX
    -d 1 -o PATH_TO_OUTPUT
```

   The sample scripts will pause while you complete this step. Once you've run get_barcodes.py, go back into Matlab and press and key to continue with the example scripts.

6. Call quantify_stripes.m or quantify_spots.m.

   The function inputs to quantify_stripes.m are the 2-d cell coordinates from step 1, the path to the text file of persistence diagrams from step 5, x- and y-boundaries of the domain (boundaryX(time_pt) and boundaryY(time_pt)), a 3-d array of the dense

xanthophore cell coordinates for all time (cellsXc), a vector containing the number of dense xanthophores for all time (numXanc), and a vector of the y-boundaries for all time (boundaryY). With the exception of the first two inputs, all other function input data are accessible from the saved .mat files of the model simulations and you do not need to generate them yourself. The last three function inputs are used for detecting when the second and third interstripes form.

The function inputs to quantify_spots.m are the cell coordinates from step 1, the path to the text file of persistence diagrams from step 5, x- and y-boundaries of the domain (boundaryX(time_pt) and boundaryY(time_pt)), the persistence threshold, and the main cell type to be used for pattern quantification. For *shady* data, we recommend using the melanophore cell data for pattern quantification (specify as input using 'M') with a persistence threshold of 90. For *pfeffer* or *nacre* data, we recommend using the iridophore cell data for pattern quantification (specify as input using 'I') with a persistence threshold of 100.

At this point, you should have successfully been able to run the program! In the following sections we review the code in more detail. The Github page also contains a helpful README and the code itself is documented.

2 Overview of the code

2.1 Main scripts

quantifying_stripes.m — This is a MATLAB function which reads in files containing cell coordinate data and corresponding dimension 1 persistence diagrams and outputs pattern statistics of zebrafish stripes. The inputs for this function are coordinate data of cell locations for the melanophores, loose iridophores, dense xanthophores, and loose xanthophores at a single time of interest ($N \times 2$ arrays, where N varies per cell type), paths to saved persistence diagram files (Ripser outputs) for the melanophores, dense xanthophores, and loose xanthophores, the x- (right) and y-boundary (top) of the domain at the time of interest (assuming lower left corner of the domain is is (0,0)), a $N_{max} \times 2 \times T$ matrix of coordinate data containing the cell locations of the dense xanthophores over time, a $T \times 1$ vector containing the number of dense xanthophore cells over time, and a $1 \times T$ vector of the y-boundaries of the growing domain over time.

The function outputs include a complete collection of pattern statistics for zebrafish stripe patterns based on topological data analysis, machine learning, and direct calculations. Specifically, quantifying_stripes.m outputs estimates for the number of stripes, number of interstripes, number of stripe breaks, number of interstripe breaks, time point when interstripes 2 and 3 begin to form, stripe width, interstripe width, mean stripe curviness, median stripe curviness, mean cell-cell spacing, variance cell-cell spacing, melanophore coefficient of variation, loose xanthophore-melanophore density counts, melanophore-loose xanthophore density counts, and loose iridophore-melanophore density counts.

quantifying_spots.m — This is a MATLAB function which reads in files containing cell coordinate data and corresponding dimension 0 persistence diagrams and outputs pattern statistics of zebrafish spots. The inputs for this function are coordinate data of cell locations for the melanophores, loose cell iridophores, dense xanthophores, and loose xanthophores at a single time of

interest ($N \times 2$ arrays, where N varies per cell type), the file path to the saved persistence diagram file (Ripser output) for a single cell of interest, the x- (right) and y-boundary (top) of the domain at the time of interest (assuming lower left corner of the domain is is (0,0)), the persistence cut for counting β_0, and the cell type to be used to compute spot statistics (options are 'M' for melanophores or 'I' for iridophores).

The function outputs include a complete collection of pattern statistics for zebrafish spot patterns based on topological data analysis, machine learning, and direct calculations. Specifically, quantifying_spots.m outputs estimates for number of spots, spot size, roundness score of spots, center stripe (X0) radius, spot alignment score, mean cell-cell spacing, variance cell-cell spacing, and the melanophore coefficient of variation.

2.2 Generating input data

Here we detail how to generate inputs for the two main scripts: 'quantify_spots.m' and 'quantify_stripes.m. We will describe how to do this specifically for simulated data from the agent-based model of A.V. and B.S., and more generally when you have arbitrary cell coordinates.

A.V.B.S. Simulation data:

1. Load in data:

```
load(PATH_TO_MAT_FILE)
```

2. Extract cell coordinates ($N \times 2$ arrays) at given time point (time_pt is specified by the user), removing cells near the boundary.

```
cutoff =  0.1*boundaryY(time_pt);
cells_mel = cellsM(find(cellsM(1:numMel(time_pt),
    2,time_pt)> cutoff &  cellsM(1:numMel(time_pt), 2,time_pt)
    < boundaryY(time_pt) - cutoff), :, time_pt);
cells_iriL = cellsIl(find(cellsIl(1:numIril(time_pt),
    2,time_pt) > cutoff &  cellsIl(1:numIril(time_pt), 2,time_pt)
    < boundaryY(time_pt) - cutoff),:, time_pt);
cells_iriD = cellsId(find(cellsId(1:numIrid(time_pt),
    2,time_pt) > cutoff &  cellsId(1:numIrid(time_pt),
    2,time_pt) < boundaryY(time_pt) - cutoff),:, time_pt);
cells_xanD = cellsXc(find(cellsXc(1:numXanc(time_pt),
    2,time_pt) > cutoff &  cellsXc(1:numXanc(time_pt),
    2,time_pt) < boundaryY(time_pt) - cutoff),:, time_pt);
cells_xanL = cellsXsn(find(cellsXsn(1:numXansn(time_pt),
    2,time_pt) > cutoff &  cellsXsn(1:numXansn(time_pt),
    2,time_pt) < boundaryY(time_pt) - cutoff),:, time_pt);
```

   Note, the variables on the right hand side of the above equations should already be saved in the .mat file from the model simulation, so you don't have to worry about generating or computing them.

3. Compute distance matrices, using periodic boundary conditions in the x-direction

```
[D_mel, ~, ~] = getPeriodicDistMats(cells_mel,
    boundaryX(time_pt));
[D_xanD, ~, ~] = getPeriodicDistMats(cells_xanD,
    boundaryX(time_pt));
```

```
[D_xanL, ~, ~] = getPeriodicDistMats(cells_xanL,
    boundaryX(time_pt));
[D_iriL, ~, ~] = getPeriodicDistMats(cells_iriL,
    boundaryX(time_pt));
```

4. Save distance matrices as text files and run Ripser in python.

```
$ python3 get_barcodes.py -i PATH_TO_DISTANCE_MATRIX
    -d 1 -o PATH_TO_OUTPUT
```

General coordinate data: Given any cell coordinate data (cells_mel, cells_iriL, cells_iriD, cells_xanD, cells_xanL), repeat steps 3-4 from above to get the corresponding persistence diagrams. The cell coordinate data should always be an $N \times 2$ array, where N is the number of cells of a given cell type. If you don't know the x-boundary for the distance matrix computation, you can set it to the maximum x-coordinate across all your cell coordinate data.

2.3 Running the program

quantifying_stripes.m In Matlab, run:

```
[num_stripes, num_Istripes, stripe_breaks, ...
Istripe_breaks, dayOfNewStripes, stripe_width,...
Istripe_width, avg_straightness, med_straightness,  ...
mean_mel_space, var_mel_space, mean_xanD_space,...
var_xanD_space, mean_xanL_space, var_xanL_space, ...
mean_melxanD_space, var_melxanD_space,  ...
mean_melxanL_space, var_melxanL_space, melCV, ...
xanL_mel_density, mel_xanL_density, iriLMel_density] ...
= quantify_stripes(cells_mel, cells_iriL, cells_xanD, ...
cells_xanL, mel1PD_dir, xanD1PD_dir, xanL1PD_dir, boundaryX,...
boundaryY, cellsXd_all, numXand_all, boundaryY_all)
```

The variable 'cellsXd_all' is the matrix of coordinate data of cell locations of the dense xanthophores over time (cellsXd_all = cellsXc), 'numXand_all' is the vector containing the number of dense xanthophore cells over time (numXand_all = numXanc), and 'boundaryY_all' is the vector of the y-boundaries of the growing domain over time. If this time series data is not available, insert dummy variables and ignore the 'dayOfNewStripes' output which estimates the first day when interstripes X1V and X1D begin to form.

quantifying_spots.m In Matlab, run:

```
[num_spots, spot_size, roundness_score, center_stripe_rad, ...
alignment_score, xanL_mel_density, mel_xanL_density, ...
mean_mel_space,  var_mel_space, mean_melxanD_space, ...
var_melxanD_space, melCV, mean_xanD_space, var_xanD_space] ...
= quantify_spots(cells_mel, cells_iriL, cells_xanD,...
cells_xanL, PD_dir, boundaryX, boundaryY, pers_cutoff, cell_type)
```

Recall, cell_type should either be 'M' or 'I' and pers_cutoff is the user-defined persistence threshold for dimension 0.

2.4 Utility scripts

Here we briefly review the Matlab functions in 'src/python' and 'src/matlab/util'.

get_barcodes.py This is a short Python function in 'src/python' that calls the ripser library

to compute persistence diagrams, given distance matrix inputs. The persistence diagram data is saved in a text file, whose save path specified by the user. To run the code and compute dimension 0 and 1 persistence diagrams:

```
$ python3 get_barcodes.py -i PATH_TO_DISTANCE_MATRIX
    -d 1 -o PATH_TO_OUTPUT
```

The -d flag specifies the maximum homology dimension for the ripser computation. For more help:

```
$ python3 get_barcodes.py -h
```

getPeriodicDistMats.m — This is a Matlab function in 'src/matlab/util' which takes as input a collection of (x,y) coordinates and computes the pairwise distances between all coordinates, with periodic boundary conditions imposed in the x-direction (horizontal direction). The periodicity condition is imposed with the boundaryX input parameter, specifying the maximum x-coordinate (and we assume the minimum x-coordinate is 0).

The inputs of this function are the coordinate data ($N \times 2$ array) and the x-boundary (boundaryX). The function outputs are the $N \times N$ distance matrix of pairwise between the input coordinates, the mean nearest-neighbor distance between the input coordinates, and the variance of the nearest-neighbor distances between the input coordinates.

my_dist_*.m — This is a collection Matlab functions in 'src/matlab/util' that computes distances between two input points, with periodic boundary conditions imposed in the x-direction (horizontal direction). Since the x-boundaries vary across fish types, we have different distance functions for the wild-type and the mutants.

The inputs to these functions are the x- and y-coordinates of any two points (cell locations), and the output is the distance between those input points (cells).

straightness_measure.m — This is a Matlab function in 'src/matlab/util' that uses topological data analysis and hierarchical clustering to compute a straightness measure of stripe patterns. Stripe straightness is defined as the percent increase in arc length distances of the stripe boundaries, in comparison to perfectly straight stripe boundaries connecting the outer most boundary cells. Stripe boundaries are detected by applying single linkage clustering the dense xanthophore cells with the number of expected clusters set to the number of expected stripes (3) minus the number of stripe breaks, as indicated by the betti numbers. Then, we take the maximum and minimum y-coordinates of each stripe cluster along a discretized x-grid to obtain upper and lower, respectively, stripe boundaries. For more details on the algorithm, see our paper.

The inputs to this function are the coordinates of the dense xanthophore cells, x-coordinates of grid points to be used to detect the stripe boundaries, the first betti numbers of the loose xanthophore and melanophore cells, a string save name for the plot, and a flag for plotting patterns with detected stripe boundaries (= 1 to plot, =0 to not plot).

This function outputs the straightness measures for the top and bottom boundary of each stripe based on the input data.

find_day_of_new_stripes.m — This is a Matlab function in 'src/matlab/util' that estimates the first

day when interstripes X1V and X1D begin to form. The code assumes interstripes X1V and X1D begin to form somewhere between 32 and 62 dpf. The estimated time at which interstripes X1V and X1D begin to form is computed by stepping through the time series of cell data between 32 and 62 dpf and identifying the first time point at which the minimum/maximum y-coordinate of the cells increases by more than 200 μm from the previous day.

The inputs to this function are a 3-d matrix of coordinate data of cell locations of the dense xanthophores over time, a vector containing the number of dense xanthophore cells over time, and a vector of the y-boundaries of the growing domain over time. The function output is the estimated day when interstripes X1V and X1D begin to form.

3 Modifying the code for real data

When working with real data or data from different types of model simulations, it may not be feasible to generate or obtain all of the input data required by this code. Here we provide tips and tricks for how to modify the code in these scenarios.

Estimating the domain boundaries If you don't know the x-boundary or y-boundaries of the domain, you can set them to the maximum of the x-coordinates and maximum of the y-coordinates, respectively, across all your cell coordinate data.

Adjusting the persistence thresholds Depending on the way in which your cell data was extracted, it might be necessary to adjust the persistence thresholds for more accurate results. We suggest fine tuning these manually by looking at a few example images first. For example, if your cells are less densely distributed, you might need to increase the persistence thresholds.

Missing time series data If time series data is not available, insert dummy variables and ignore the 'dayOfNewStripes' output which estimates the first day when interstripes X1V and X1D begin to form.

Only having data for one cell type If you only have data for one or two types of pigment cells, you can adjust the code to only compute statistics pertaining to that cell type. Be careful to note that some statistics of one cell type might rely on information of a different cell type.

Data with both spots and stripes If you have a lot of images/simulations to analyze, you may have both striped and spotted patterns. In this case, you probably don't want to have to manually split the images in groups and predetermine which should be inputs to quantifying_stripes.m versus quantifying_spots.m. Instead, you can compute the dimension 0 and dimension 1 persistence diagrams and use the corresponding betti numbers to determine if the image at hand is a spotted or striped pattern. If $\beta_1 \approx 0$ and $\beta_0 \gg 0$, assume spots and use quantifying_spots.m. If $\beta_0 \approx \beta_1$, assume stripes. Then, call the appropriate Matlab script to complete the pattern quantification and use quantifying_stripes.m.