
A WEB-BASED APPLICATION OF THE CELLULAR
FORCE-INFERENCe TOOLKIT (CELLFIT)

HONORS THESIS
SUBMITTED TO THE FACULTY OF THE
DEPARTMENT OF PHYSICS AND ASTRONOMY, VANDERBILT UNIVERSITY
APRIL, 2019

BY

XIAOJIA J. XU

ADVISOR:
PROFESSOR M. SHANE HUTSON

COMMITTEE OF EXAMINERS:
PROFESSOR KÁLMÁN VARGA
PROFESSOR JOHN WIKSWO
PROFESSOR ALFREDO GURROLA



VANDERBILT
UNIVERSITY

ACKNOWLEDGMENTS

I extend my gratitude to my thesis advisor, Prof. Shane Hutson, for his consistent guidance throughout the year. Thanks to Tyler McCleery for helping me get started with the project. Thanks to James O'Connor for providing the data from his experiments. Thanks to the committee members for taking time to consider this thesis. Finally, thanks to the Department of Physics and Astronomy and Vanderbilt for providing the opportunities to write this thesis.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	i
LIST OF ACRONYMS	iii
LIST OF FIGURES	iv
I Introduction	1
I.1 Active matter physics and wound healing	1
I.2 Image based inference of cellular forces in epithelial tissues	3
I.3 Motivation to redesign CellFIT	3
II Software	4
II.1 Image processing	4
II.1.1 Image segmentation	4
II.1.2 Mesh generation	5
II.1.3 Angle and curvature determination	7
II.2 Calculations	8
II.2.1 Formulating and solving tension equations	8
II.2.2 Formulating and solving pressure equations	11
II.3 Web-based application	13
III Results and Discussion	14
III.1 Validation using synthetic data	14
III.2 Inferring cellular forces in <i>Drosophila</i> epithelia	14
III.3 Discussion	15
III.4 Conclusion	15
REFERENCES	19

LIST OF ACRONYMS

CellFIT: The Cellular Force-Inference Toolkit

PyCellFIT: Python implementation of CellFIT

TJ: Triple junction

GDAL: Geospatial Data Abstraction Library

FEM: Finite element model

LIST OF FIGURES

Figure	Page	
I.1	Wounds induced in Madin-Darby canine kidney epithelial cell monolayers. Scale bar represents $50 \mu m$	1
I.2	Time-lapse series of wound healing in <i>Drosophila</i> embryonic tissue.	2
II.1	Image of a <i>Drosophila</i> embryonic epithelial wound taken 10 minutes after laser ablation. Image credit: James O'Connor.	5
II.2	A zoomed in section of a TJ in a watershed image.	6
II.3	Mesh generated from the watershed image in Figure II.1 (b)	7
II.4	Illustration from Brodland et al. (2014) to show angles at a TJ and curvature at a boundary.	8
II.5	Overdetermined tension equations.	9
III.1	90 degree honeycomb lattice	14
III.2	150 degree honeycomb lattice	15
III.3	Tension and pressure solutions of a synthetic mesh.	16
III.4	Raster image of a mature wing of the late <i>Drosophila</i> pupa.	16
III.5	Tension and pressure solutions of a mesh from the mature wing of the late <i>Drosophila</i> pupa.	17
III.6	Mesh of a <i>Drosophila</i> embryonic epithelial wound taken 10 minutes after laser ablation.. . . .	17
III.7	Image of a <i>Drosophila</i> embryonic epithelial wound taken two hours after laser ablation. Image credit: James O'Connor.	18

CHAPTER I

Introduction

I.1 Active matter physics and wound healing

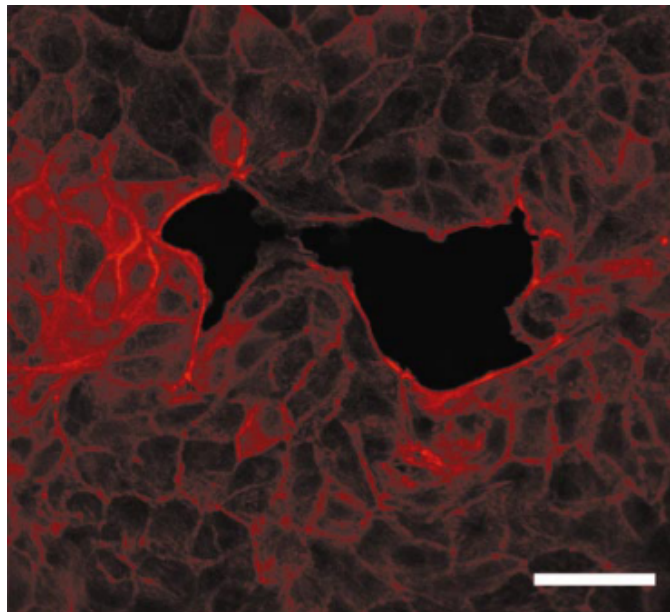


Figure I.1: Wounds induced in Madin-Darby canine kidney epithelial cell monolayers. Scale bar represents $50 \mu m$.

Sriram Ramaswamy FRS, who helped to found the field of active matter, stated that biology is increasingly concerned with the mechanical forces exerted in living systems, and the active-matter approach provides the framework in which to ask these questions. It has made specific predictions, and experimenters are managing to check some of them. The impact on biology is already being felt, from the subcellular scale to starling flocks. In cell biology, tissue is clearly the next mountain, and some steps on it have already been taken [13]. Physicists now understand that there is new physics in living matter, and that there is no better laboratory than the living cell to study the physics of systems far from thermodynamic equilibrium [16]. Historically, there was little focus on working out the principles of self-organization until the mid-1990s [12]. In 1995, Tams Vicsek proposed a flocking model that reproduced the movements of bird flocks and fish schools [18]. Early experiments in 1997 showed that life like structures could self-assemble from microtubules and a few proteins supplied with ATP [14]. These early attempts to understand self-organization in active

matter systems sparked a growing number of studies in the past two decades. Initially, the number of simulations grew rapidly but the number of quantitative experiments could not keep up. There exists a large number of papers that provide image data of wound healing processes, for example, the one showed in Figure I.1 [4]. However, there yet exists no complete description of tissue-level mechanics.

Physicists and biologists have long collaborated to develop quantitative tools to study living systems, but they have also had their differences in approach. For example, some physicists may consider living systems to be too complex to be captured by a set of general principles. On the other hand, biologists may consider physical approaches to be too abstract to capture, with specificity, the wide range of phenomena in cells and tissues.

The focus of this thesis paper is on a tool that is applied to studying the living systems in *Drosophila*, particularly its embryonic tissues where cells are capable of quickly generating coordinated forces that close small wounds without a scar. The possibility that the understanding of wound healing in *Drosophila* embryonic tissues could translate to other epithelial systems provides great motivation for the study [8]. Figure I.2 is an example of a wound closure time-series. [2].

As of now, wound healing is understood to have four distinct processes: expansion, coalescence, contraction and closure [2]. Cell edge tensions are created during these processes and these tensions are assumed to arise from the combined action of actomyosin contractions, membrane tensions and cell-cell adhesion systems. The original goal of the thesis is to attempt to apply CellFIT to observe the spatial and temporal variations in the tension throughout the cell layer. However, given the time constraint on a senior thesis project, the goal is limited to developing an improved Python implementation of CellFIT.

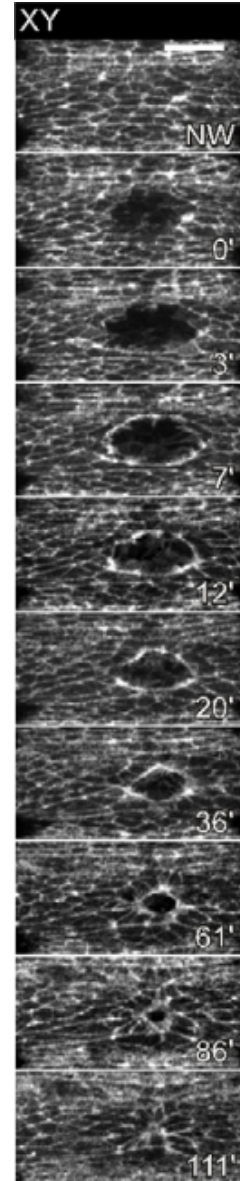


Figure I.2: Time-lapse series of wound healing in *Drosophila* embryonic tissue.

I.2 Image based inference of cellular forces in epithelial tissues

The collection of cellular forces generated in epithelial tissues give rise to crucial biological processes, such as wound healing and embryogenesis. To study these tissue-scale processes, we can estimate the cell-cell interface tensions by using image-based inference tools. CellFIT is a package of force inference equations and assessment tools that calculates the forces in cell edges based on their geometries [3][17]. Given an image of an epithelial cell sheet, CellFIT can infer cellular forces by segmenting the image into individual cells, extracting information from the observable curvature of cell interfaces, constructing equilibrium equations for the points where cells meet at triple junctions, and finding a least-squares solution for the tensions at cell-cell interfaces. Similarly, cellular pressures can be estimated by constructing Laplace equations that relate the edge tensions, curvatures and cellular pressure differences.

I.3 Motivation to redesign CellFIT

The previous implementation of CellFIT in C++ has demonstrated its functionalities through a graphical user interface on the Windows Operating System. Despite these capabilities, the accessibility of CellFIT to scientists of all backgrounds is not yet optimized. As mentioned in the previous section, the main focus of the thesis is to first develop a Python implementation of CellFIT and then create a web-based application of CellFIT that allows users to easily access the software from a browser. With this approach, all scientists can easily access CellFIT from any operating system platform. Many users of the current C++ implementation of CellFIT reported 'mysterious' software crashes that prevented partial or full analysis of their data. The updated Python implementation would include improved user interface, error handling and the implementation of additional functionality for reading and processing image stacks, such as Figure I.2. The new Python implementation of CellFIT will be referred to as PyCellFIT for simplicity.

CHAPTER II

Software

This chapter describes the methods used in redesigning CellFIT using Python, a dynamic interpreted language with high readability and compact modularity. The Python implementation capitalizes on the extensibility of Python by heavily utilizing the NumPy library, which contains high performing functions for numerical calculations [10][11]. The bulk of the calculations are carried out through NumPy and thus reducing execution time. Consider a benchmark test done on the matrix multiplication of two random matrices $A, B \in^{2000 \times 2000}$ [15].

NumPy (1.3.0)	120 seconds
Python (2.6.5)	3360 seconds
C++ (gcc 4.4.3)	27 seconds
Java (1.6.0_20)	1620 seconds

Table II.1: Benchmark time of tests ran on Acer TravelMate 5735Z; 2x Pentium(R) Dual-Core CPU T4500 @2.30GHz; RAM 4 GB; Intel GMA 4500MHD; Ubuntu 10.10.04 LTS. Jobs are ran on a single thread.

The Python, Java and C++ tests use the ijk-algorithm while the built in NumPy function is used for the Numpy test. Although NumPy is about four times slower than C++, the NumPy least-squares calculations performed in this study is on the order of 10^{-2} seconds. For instance, the tension calculations for the mesh of a mature wing of the late *Drosophila* pupa shown in section III.2 take $1.34 \text{ ms} \pm 112 \text{ } \mu\text{s}$ per loop (mean \pm std. dev. of 7 runs, 1000 loops each). Therefore, PyCellFIT experiences no significant slow down by using the NumPy library instead of a C++ algorithm.

II.1 Image processing

If an image has already been processed with the C++ CellFIT, then there should be an existing finite element model (FEM) stored in a text file. In that case, the steps described in this section may be skipped and calculations can be directly be performed with the FEM file.

II.1.1 Image segmentation

The first step in CellFIT is to segment raw images into watershed images with Seedwater Segmenter [9] or another segmentation software with equal capabilities. The segmentation is done by using a

watershed algorithm, which can be briefly explained by the analogy of filling water in a valley until the outline of the valley is shown by the edge of the water. An example is shown in Figure II.1.

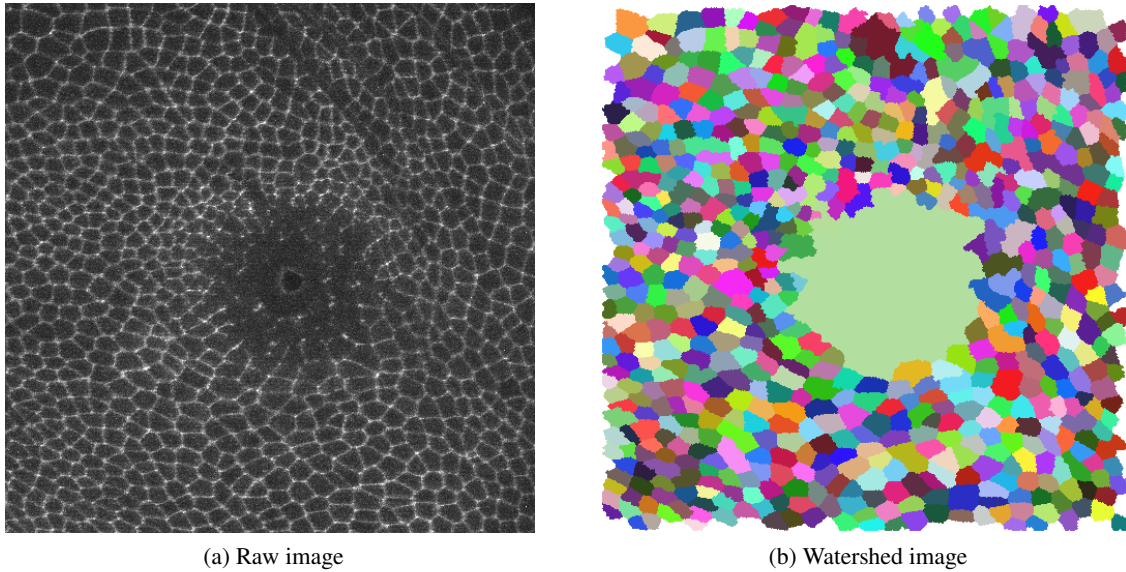


Figure II.1: Image of a *Drosophila* embryonic epithelial wound taken 10 minutes after laser ablation. Image credit: James O'Connor.

There certainly exist cases where an user has already segmented their raw image with a different method. Fortunately, the Seedwater Segmenter can be simply reapplied to the already segmented image to ensure that the mesh generation can be carried out without errors.

II.1.2 Mesh generation

This step involves converting the raster-based description to a vector-based description. creating a mesh consisted of nodes, edges and cells from the watershed image. A given cell in the watershed image is constituted of pixels with identical values, which are used to assign cell IDs. Depending on the bit size of watershed image, the pixel values may be RGB or grayscale. A Python Geospatial Data Abstraction Library (GDAL) [6] is imported for cost-effectively converting the watershed image into a NumPy array, which can easily manipulated. A downside of this short cut is that in some cases, cells with different RGB values will end up with the same grayscale value due to GDAL's conversion process. An additional error checking function would be useful in preventing such mistakes.

Once the image data is stored in a NumPy Array, the TJs can be first identified. Refer to Figure

II.2 (a) to see that each node in the watershed image pixel grid has four neighboring pixels. The integers 1,2,3 represent the cell IDs. The TJ, marked by the circular red dot, in the image can be located by finding the node with three unique neighboring pixel values.

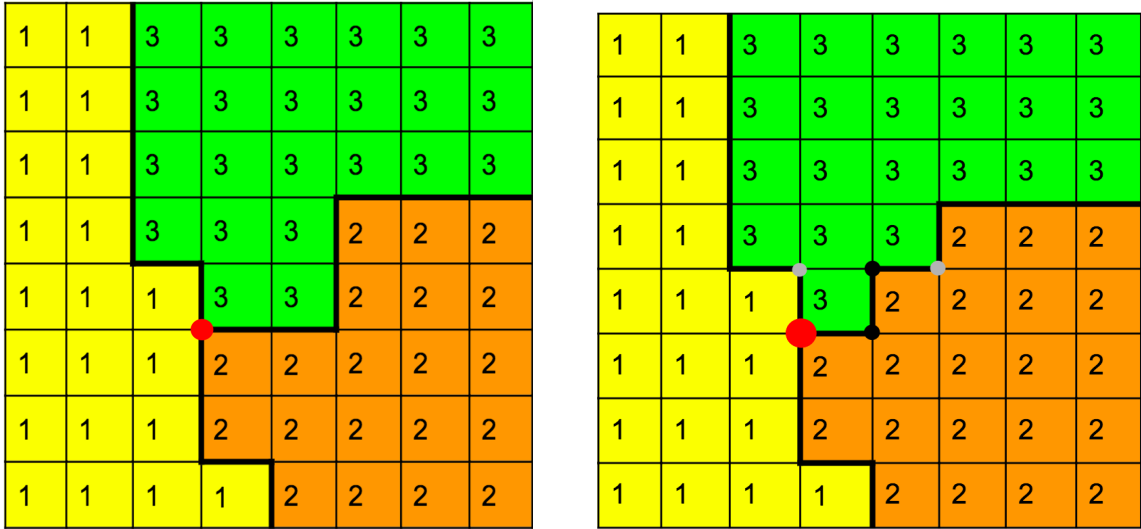


Figure II.2: A zoomed in section of a TJ in a watershed image.

By scanning the entire array, we can quickly obtain the coordinates of the TJs. The next step is to create the edges by tracing along the nodes that have two unique neighbours. It is also important to keep track of the two neighboring cells as the edge is traced, otherwise the edge can go off the intended track. See Figure III.2 (b) for an example where this error may occur. Starting from the red dotted TJ, we can trace the edge going to the right by recording the nodes, marked by black dots, that make up the edge. However, there are two valid nodes marked by grey dots to continue the edge but the grey dot on the right is the only correct option. By keeping track of cell IDs 2 and 3, we can make sure that we are following the correct edge. For each edge, the following information has to be recorded: an unique edge ID; starting and ending TJs; and the intermediate edge nodes.

The number of intermediate nodes will determine the method of fitting the edge afterwards. There are three methods for fitting the edge: chord fit; near segment fit; and circle fit. Chord fit involves connecting all the TJs and forming polygon-shaped cells. This is the simplest fit and no

intermediate edge nodes are required. Near segment fit requires connecting a given number of intermediate edge nodes, specified by the user. Lastly, circle fit, as the name suggests, creates a circular arc that most suitably fits all the intermediate edge nodes associated with a given edge.

After the mesh generation is completed, a FEM file can be saved and the mesh can be visualized by plotting all the nodes and edges. See Figure II.3 for an example of a completed mesh.

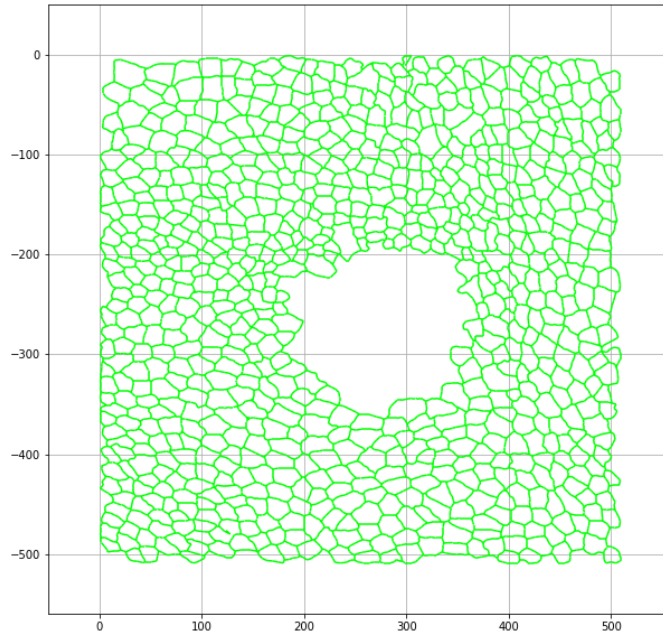


Figure II.3: Mesh generated from the watershed image in Figure II.1 (b)

II.1.3 Angle and curvature determination

An important step in extracting information from the images is measuring the observable angles and curvatures of the cell interfaces. There are two main methods to determine angles: nearest segment and circular fit. The former is not suitable for cells with more circular edges but it is good for cells with crenulated edges. The latter is generally the best approach for most cases, as argued by Veldhuis et al. (2015). The angles are determined by measuring the angle of each circular arc at the point where it makes its closest approach to the TJ. For example, the vector for the tension γ_j is drawn in Figure II.4 to show the angle with respect to the horizontal axis they make at their closest approach to the TJ labelled B. These angles are very important as they are needed to write equilibrium equations at each TJ, which are used to calculate the edge tensions in the system. See section II.2.1 for more details.

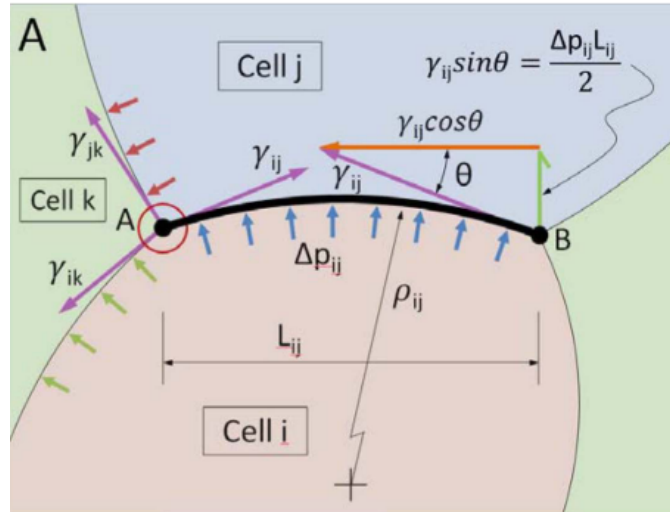


Figure II.4: Illustration from Brodland et al. (2014) to show angles at a TJ and curvature at a boundary.

Since the radii of curvature are required for writing down Laplace equations to calculate pressures, the curvatures must be determined at this step if the user chooses to find pressure solutions. If the circle fit was used previously, then the radius of curvature can be easily calculated from the circular arc used for fitting. An example of a radius of curvature of a cell-cell boundary is ρ_{ij} shown in Figure II.4. PyCellFIT currently do not have circular fit functionalities so curvatures must be obtained from C++ CellFIT before making pressure calculations.

II.2 Calculations

If a preexisting FEM file is present, then PyCellFIT can directly read in the file and load the model into memory for analysis. This capability reduces the hassle of regenerating the mesh every time the same image is used for analysis.

II.2.1 Formulating and solving tension equations

Let us define the edge tension as the net contractile force along a given cell edge. The tension, denoted as γ , along the edges of cells are assumed to be uniform within the given edge. Cell shape changes are assumed to undergo slow transitions, such that the cells can be considered quasi-static. This assumption demands that all triple junctions (TJ) are in equilibrium, thus each triple junction needs to satisfy the force balance equation below.

$$\sum \gamma_{mn} \hat{\mathbf{r}}_{mnA} = 0 \quad (\text{II.1})$$

where the unit vectors $\hat{\mathbf{r}}_{mnA}$ are constructed tangent to the limiting angle at which the membrane along the boundary between cells m and n approaches the A^{th} triple junction and pointing away from the junction, and the summation is carried out over all edges that connect to that TJ. For each TJ, two equations can be written by resolving in the x- and y- directions. The three tensions for a given TJ cannot be solved because there are only two equations. However, as more TJs are considered in the system, the system becomes overdetermined, shown by Figure II.1 [3].

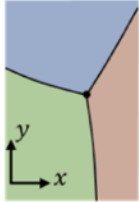
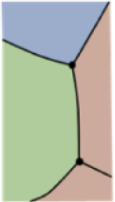


Representative Configuration				
No. of Equations	2	4	12	42
No. of Unknowns	3	5	12	37

Figure II.5: Overdetermined tension equations.

We can then write the tension equations in matrix form. Let us use a $N_{TensionEqns}$ by $N_{Tensions}$ matrix \mathbf{G}_γ to hold the sines and cosines of all the equations.

$$\mathbf{G}_\gamma \boldsymbol{\gamma} = \mathbf{0} \quad (\text{II.2})$$

Equation II.2 is homogeneous and yields the trivial solution $\boldsymbol{\gamma} = \mathbf{0}$. We can get around this issue by utilizing a constrained least-squares equation system. We construct and solve a constrained least-squares equation by first populating a matrix \mathbf{G}_γ associated with the tensions.

$$\begin{bmatrix} \mathbf{G}_\gamma^T \mathbf{G}_\gamma & \mathbf{C}_1^T \\ \mathbf{C}_1 & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \gamma_1 \\ \vdots \\ \gamma_{N_{Tensions}} \\ \lambda_1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ \vdots \\ 0 \\ N_{Tensions} \end{Bmatrix} \quad (\text{II.3})$$

where

$$\mathbf{C}_1 = \{1 \dots 1\} \quad (\text{II.4})$$

\mathbf{C}_1 imposes the constraint that the average tension $\bar{\gamma} = 1$ and λ_1 is the associated Lagrange multiplier. The solve function from the NumPy linear algebra (`numpy.linalg.solve`) is used to solve for tensions as follows.

```
G = np.mat(np.zeros((N_Ten, 2*N_TJ)))
```

```
A = np.zeros((N_Ten+1, N_Ten+1))
```

```
b = np.zeros(N_Ten+1)
```

```
A[:-1, -1] = 1
```

```
A[-1, :-1] = 1
```

```
b[-1] = N_Ten
```

```
# go through all triple junctions
```

```
for i in range(N_TJ):
```

```
    # go through the 3 edges of each triple junction
```

```
    for vec in range(3):
```

```
        # norm of the vector representing an edge
```

```
        norm = np.sqrt(LIST_Edge_Vectors[i][2*vec]**2 + LIST_Edge_Vectors[i][2*vec+1]**2)
```

```
        # column number that correspond to the particular tension
```

```
        col = MAP_Edge_to_Tension(ID_TJ_Edges[3*i + vec])
```

```
        # assign values to matrix
```

```
        G[col, 2*i] = LIST_Edge_Vectors[i][2*vec] / norm
```

```
        G[col, 2*i + 1] = LIST_Edge_Vectors[i][2*vec + 1] / norm
```

```

C = 2*np.matmul(G, G.T)
A[: -1, : -1] = C
gamma = linalg.solve(A, b)

```

II.2.2 Formulating and solving pressure equations

In certain cases, there is not a need to determine the pressure after determining the tensions. If pressures are deemed useful, then a set of Laplace equations can be formulated for the given image. At the boundary between two cells i and j , the difference in pressure is Δp_{ij} .

$$\Delta p_{ij} = p_i - p_j \quad (\text{II.5})$$

where p_i and p_j are the respective pressures in cells i and j . Using the Laplace equation, we can relate the pressure difference to the corresponding tension and radius of curvature.

$$\Delta p_{ij} = \frac{\gamma_{ij}}{\rho_{ij}} \quad (\text{II.6})$$

The radius of curvature is positive if the boundary is convex into cell j . If we combine the above two equations, we can obtain a new equation for each boundary.

$$p_i - p_j = \frac{\gamma_{ij}}{\rho_{ij}} \quad (\text{II.7})$$

And all the equations for each boundary can be assembled into a matrix form.

$$\mathbf{G}_p \mathbf{p} = \mathbf{y} \quad (\text{II.8})$$

Each column in \mathbf{G}_p represents a cell. Each row in \mathbf{G}_p represents an equation at a boundary and contains a 1 and a -1, while the other row entries are zero. An entry of 1 indicates that the cell is pushing out and an entry of -1 indicates the opposite. The formalism provided here is different from the original CellFIT papers [3][17]. The right hand side of Equation II.8 is expressed here as \mathbf{y} because there is an additional step in constructing the right hand column vector in the constrained

least-squares system [5]. Now we can begin using another constrained least-squares equation system to acquire pressure solutions.

$$\mathbf{q} = 2\mathbf{G}_p^T \mathbf{y} \quad (\text{II.9})$$

\mathbf{q} is a column vector with length $N_{pressures}$ and an additional zero entry is appended to it to make up the right hand side of the least-squares system.

$$\begin{bmatrix} 2\mathbf{G}_p^T \mathbf{G}_p & \mathbf{C}_2^T \\ \mathbf{C}_2 & \mathbf{0} \end{bmatrix} \begin{Bmatrix} p_1 \\ \vdots \\ p_{N_{pressures}} \\ \lambda_2 \end{Bmatrix} = \begin{Bmatrix} q_1 \\ \vdots \\ q_{N_{pressures}} \\ 0 \end{Bmatrix} \quad (\text{II.10})$$

where

$$\mathbf{C}_2 = \{1 \dots 1\} \quad (\text{II.11})$$

\mathbf{C}_2 imposes the constraint that the average pressure $\bar{p} = 0$ and λ_2 is the associated Lagrange multiplier. Once the augmented matrix and right hand column are ready, the solve function from the NumPy linear algebra (`numpy.linalg.solve`) and be used to efficiently calculate pressure solutions.

```

y = np.zeros(N_Tensions)
y = gamma_p / LIST_rho
for i in range(N_Ten):
    if LIST_rho[i] < 0:
        y[i] = 0
q = np.zeros(N_Cell + 1)
q[:-1] = 2 * np.matmul(Gp.T, y)
Ap = np.zeros((N_Cell+1, N_Cell+1))
Ap[:-1, -1] = 1
Ap[-1, :-1] = 1
Cp = 2 * np.matmul(Gp.T, Gp)

```

```
Ap[: -1, : -1] = Cp  
pressures = linalg.solve(Ap, q)
```

II.3 Web-based application

The focus of the thesis project is constrained to developing PyCellFIT first before launching the web-based application. However, a toy implementation on Science Flask [7] has been made to explore the ways in which the web-based app can be most efficiently developed.

CHAPTER III

Results and Discussion

III.1 Validation using synthetic data

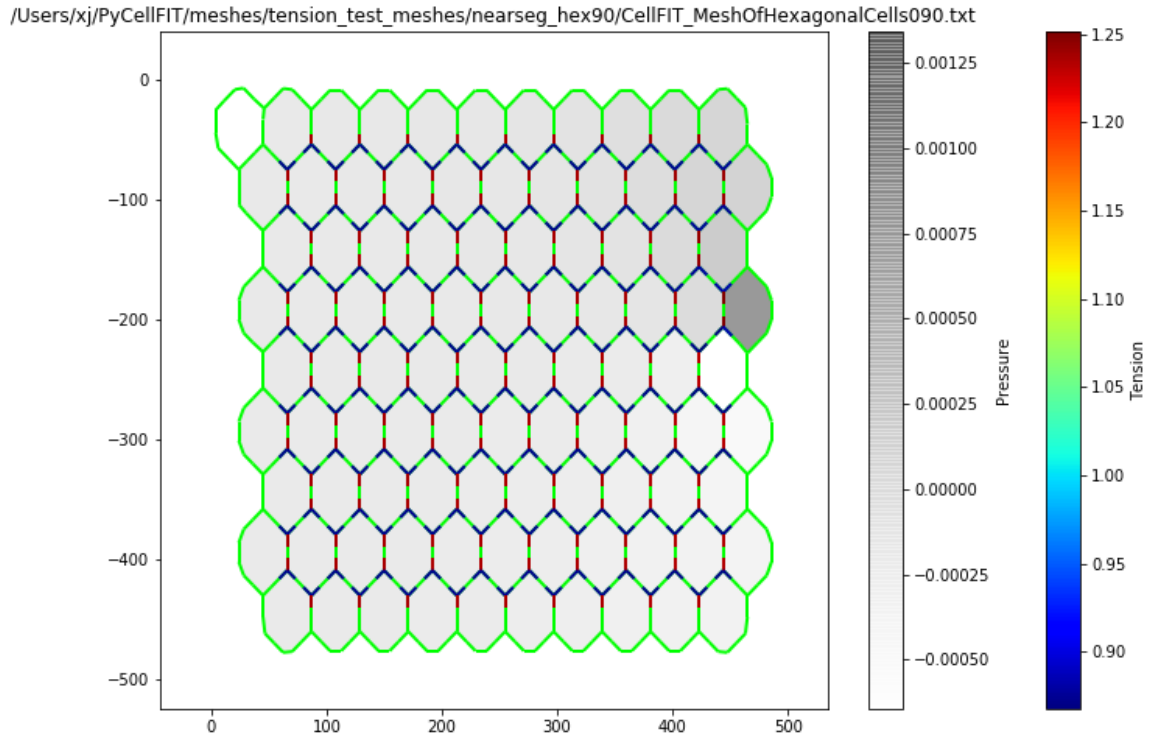


Figure III.1: 90 degree honeycomb lattice

III.2 Inferring cellular forces in *Drosophila* epithelia

In the mature wing of the late *Drosophila* pupa, cells are predominantly hexagonal and resemble a honeycomb pattern [1].

Finally, we arrive at the crux of the problem - applying CellFIT to an image of the a wound healing process. The image is from an experiment done on January 17, 2019 by James O'Connor, who is in the Page-McCaw lab in the Vanderbilt University Medical Center.

The image taken two hours after laser ablation encountered more difficulties.

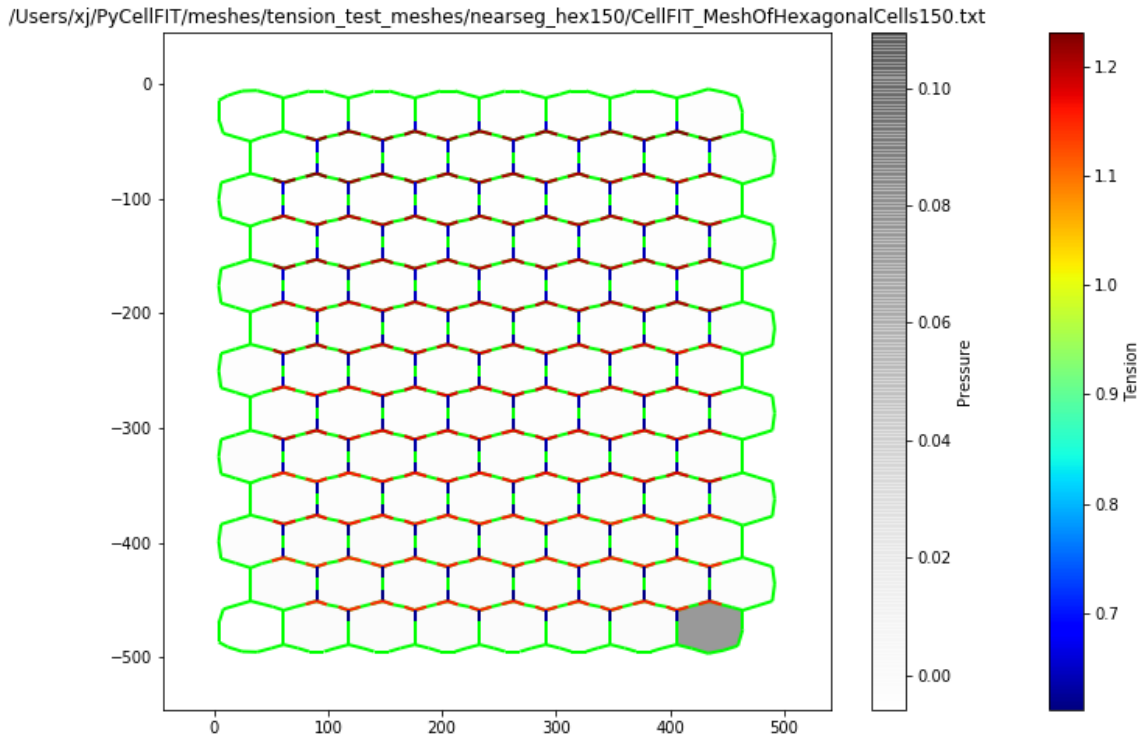


Figure III.2: 150 degree honeycomb lattice

III.3 Discussion

I hope to apply the web-based CellFIT to time-resolved image stacks of wound healing in *Drosophila* epithelia to demonstrate spatial and temporal variations in cellular forces as the wounds close.

Feeding CellFIT images from different segmentation software is observed to cause crashes.

Optical tweezers (2014), laser ablation (early 2000s), FRET (not very good).

Traction force microscopy

III.4 Conclusion

The Python implementation of CellFIT is available on GitHub at <https://xj-xu.github.io/PyCellFIT/>.

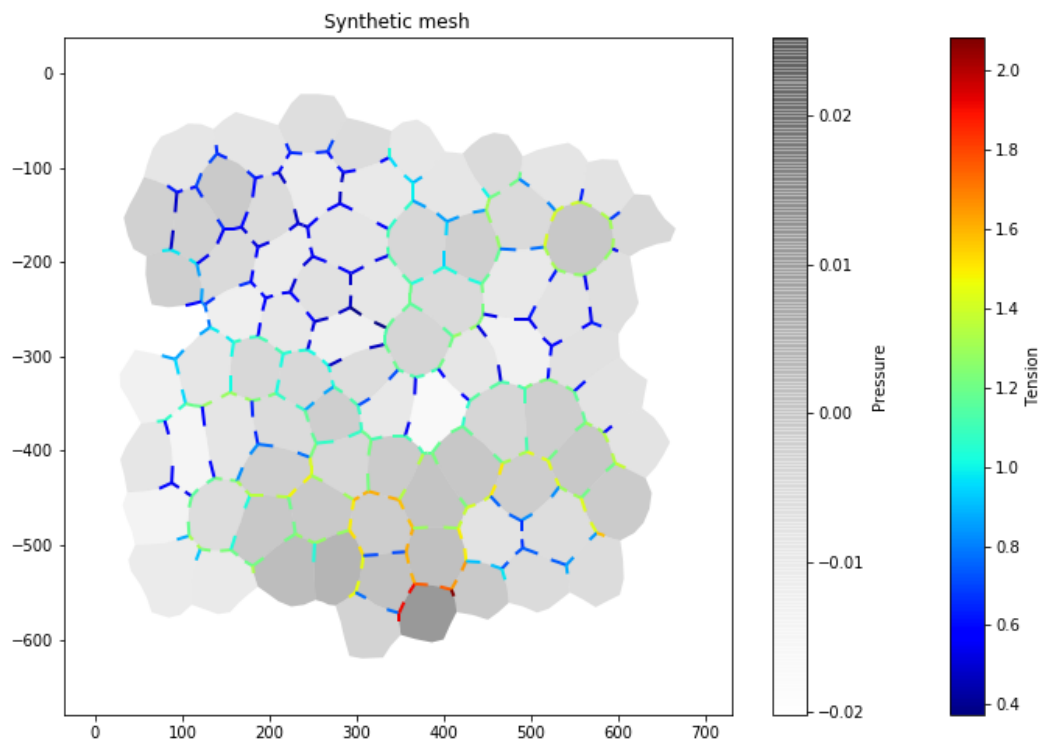


Figure III.3: Tension and pressure solutions of a synthetic mesh.

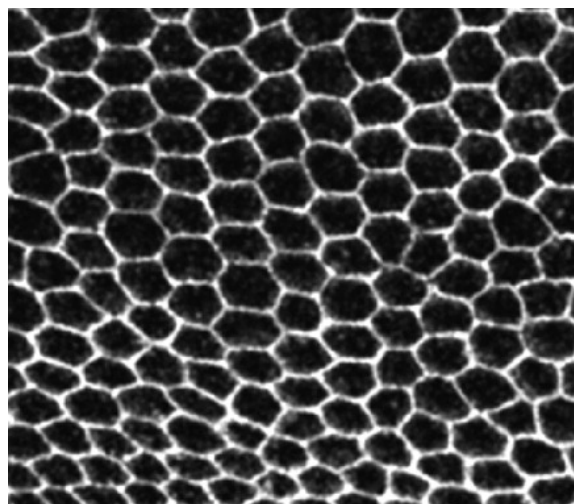


Figure III.4: Raster image of a mature wing of the late *Drosophila* pupa.

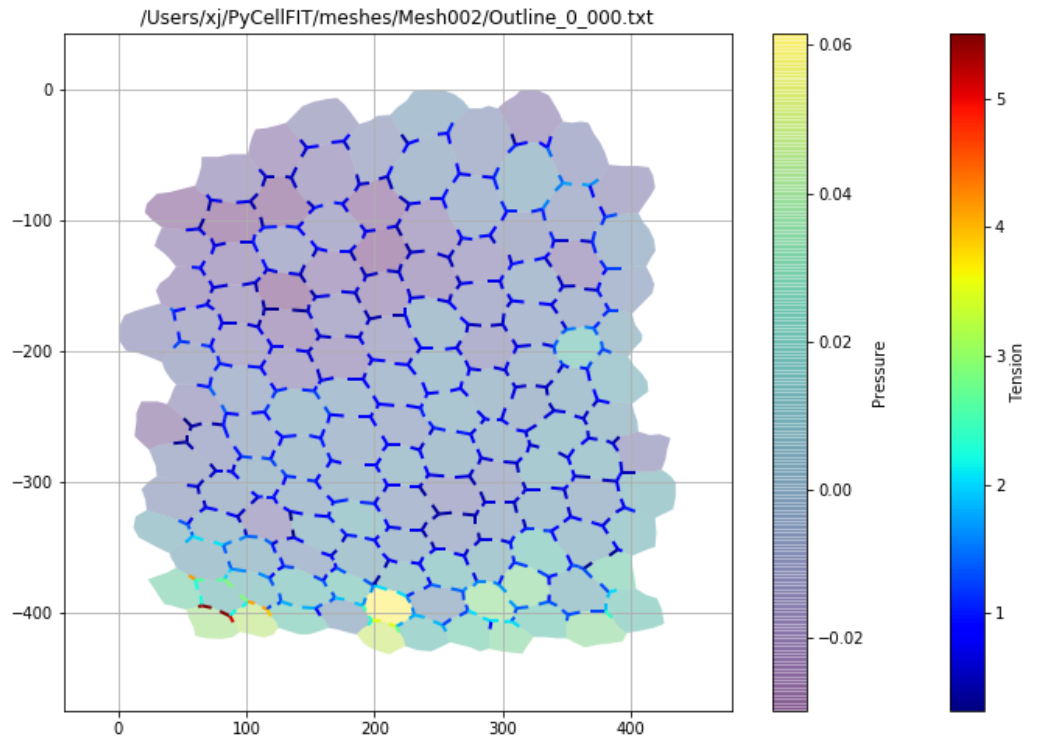


Figure III.5: Tension and pressure solutions of a mesh from the mature wing of the late *Drosophila* pupa.

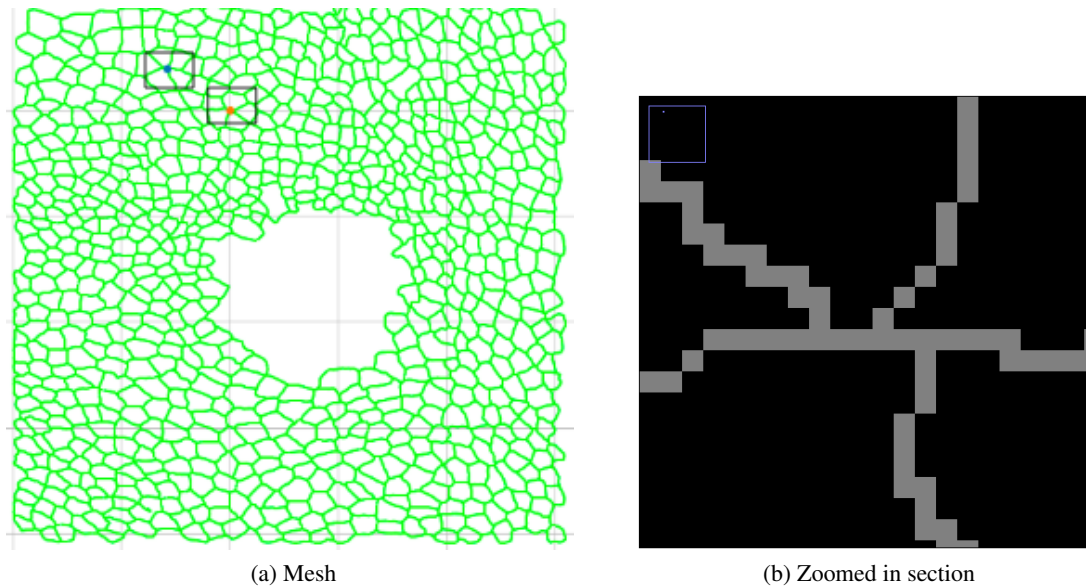
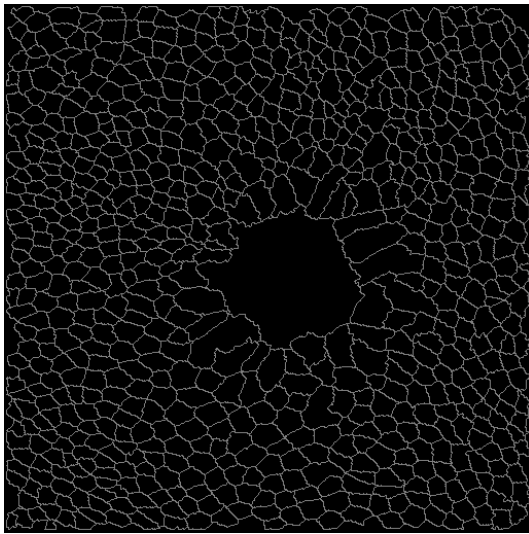


Figure III.6: Mesh of a *Drosophila* embryonic epithelial wound taken 10 minutes after laser ablation..



(a) Outline image



(b) Watershed image

Figure III.7: Image of a *Drosophila* embryonic epithelial wound taken two hours after laser ablation. Image credit: James O'Connor.

REFERENCES

- [1]Epithelial Organization: May the Force Be with You. *Current Biology*, 18(4), 2008. ISSN 09609822. doi: 10.1016/j.cub.2007.12.030.
- [2]Maria Teresa Abreu-Blanco, S. M. Parkhurst, Jeffrey M Verboon, James J Watts, and Raymond Liu. *Drosophila* embryos close epithelial wounds using a combination of cellular protrusions and an actomyosin purse string. *Journal of Cell Science*, 125(24):5984–5997, 2013. ISSN 0021-9533. doi: 10.1242/jcs.109066. URL <http://jcs.biologists.org/content/joces/125/24/5984.full.pdf>.
- [3]G W Brodland, J H Veldhuis, S Kim, M Perrone, and D Mashburn. CellFIT: A Cellular Force-Inference Toolkit Using Curvilinear Cell Boundaries. *PLoS ONE*, 9(6):99116, 2014. doi: 10.1371/journal.pone.0099116.
- [4]Gabriel Fenteany, P Janmey, and T Stossel. Signaling pathways and cell mechanics involved in wound closure by epithelial cell sheets. *Curr Biol*, 10(14):831–838, 2000.
- [5]Henri P. Gavin. Constrained linear least squares. 2015. URL <http://people.duke.edu/~hpgavin/cee201/constrained-least-squares.pdf>.
- [6]GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2018. URL <http://gdal.org>.
- [7]Daniel Homola. *Science Flask*. URL https://github.com/danielhomola/science_flask.
- [8]M. Shane Hutson. Cellular diversity heals. *Nature Physics*, 14(7):639–641, jul 2018. ISSN 1745-2473. doi: 10.1038/s41567-018-0192-y. URL <http://www.nature.com/articles/s41567-018-0192-y>.
- [9]David N. Mashburn, Holley E. Lynch, Xiaoyan Ma, and M. Shane Hutson. Enabling user-guided segmentation and tracking of surface-labeled cells in time-lapse image sets of living tissues. *Cytometry Part A*, 81 A(5):409–418, 2012. ISSN 15524922. doi: 10.1002/cyto.a.22034.
- [10]T. E. Oliphant. Python for scientific computing. *Computing in Science Engineering*, 9(3):10–20, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.58.

- [11] Travis E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015. ISBN 151730007X, 9781517300074.
- [12] Gabriel Popkin. The physics of life. *Nature*, 529(7584):16–18, 2016. ISSN 14764687. doi: 10.1038/529016a. URL <http://www.nature.com/doifinder/10.1038/529016a>.
- [13] Sriram Ramaswamy. The Mechanics and Statistics of Active Matter. *Annu. Rev. Condens. Matter Phys.*, 2010. ISSN 1947-5454. doi: 10.1146/annurev-conmatphys-070909-104101. URL <http://arxiv.org/abs/1004.1933>{%}0A<http://dx.doi.org/10.1146/annurev-conmatphys-070909-104101>.
- [14] T Surrey, A C Maggs, S Leibler, and F J Ne. Self-organization of microtubules and motors. *Nature*, 389(September):305–308, 1997.
- [15] Martin Thoma. Performance of matrix multiplication in python, java and c++, 2012. URL <https://martin-thoma.com/matrix-multiplication-python-java-cpp/>.
- [16] Xavier Trepat and Erik Sahai. Mesoscale physical principles of collective cell organization. *Nature Physics*, 14(7):671–682, 2018. ISSN 17452481. doi: 10.1038/s41567-018-0194-9. URL <https://doi.org/10.1038/s41567-018-0194-9>.
- [17] Jim H Veldhuis, David Mashburn, M. Shane Hutson, and G Wayne Brodland. Practical aspects of the cellular force inference toolkit (CellFIT). *Methods in Cell Biology*, 125:331–351, 2015. doi: 10.1016/bs.mcb.2014.10.010.
- [18] Tamas Vicsek, Andras Czirok, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel Type of Phase Transition in a System of Self-Driven Particles. *Physical Review Letters*, 75, 1995. URL <https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.75.1226>.