Assignment 3 – Cellular Structure and Dynamics

CS/BIOE/CME/BIOPHYS/BIOMEDIN 279

Due: November 29, 2016 at 3:00 PM

In the previous assignments, we've talked about the structure of biomolecules at an atomic resolution. The goal of this assignment is to discuss structure and organization at the cellular level, and to gain an appreciation for the techniques we can use to learn about these structures.

Acknowledgements: Many thanks to Prof. Julie Theriot and Prof. Zach Pincus for access to the data sets used in this assignment. For more on their methods and software, check out:
Pincus Z & Theriot JA. Comparison of quantitative methods for cell-shape analysis. J Microscopy 227 pp. 140156 (2007).

1 Preliminaries

Download and unzip the Assignment 3 Setup source file from the website. Please read the Assignment 3 Setup handout prior to starting this assignment.

2 Cellular Structure

What do we mean when we refer to cellular structure and organization? Unlike the atoms within a biomolecule, the molecules within a cell do not typically have 3D coordinates that are conserved within all individual cells of a given type. We know that the overall shape of a given cell (and the organization of bio-machinery within that cell) is strongly tied to its functionality and chemical environment.

Humans can observe the variations in cellular structure that occur across cell types, between mutants, and after changes in chemical environment. These observations make for a strong qualitative analysis, but is it possible to develop a quantitative method for analyzing the structure of cells that captures this human intuition? Modern image processing techniques and basic statistical procedures allow us to extract meaningful features of different cell types.

2.1 Microscopy Image Analysis

Before doing anything else, let's start by looking at some of the microscopy images we will be analyzing. Take a look at the images in `imgs/caulobacter` and `imgs/keratocytes`. These are grayscale images, where each pixel value represents the observed intensity at the corresponding (x, y) point. Of course, we can also use colors other than gray to represent different intensities. There are many ways to look at images, but we provide you with a simple script to display a given file.

Usage:

```
python show.py <path to image file>
```

For example, to display the image ”keratocytes/A.tif” in the ”imgs” directory. From the ”imgs” directory run

```
python show.py keratocytes/A.tif
```

In looking at these images, it should be clear that they are noisy. Let’s start by trying to denoise the keratocyte images using some of the tricks we talked about in class. In the next few exercises, you will construct various filters. To run these filters on images, use the `filter.py` script.

Usage:

```
python filter.py <path to image file> <filter name>
```

For example, to apply the ”mean” filter to the image ”keratocytes/A.tif” in the ”imgs” directory. From the ”imgs” directory run

```
python filter.py keratocytes/A.tif mean
```

Question 1: *Implement the* `mean` *low-pass filter in* `filter.py`*. This filter will convolve the original image with a constant function. Play around with the window size: try a* 3×3 *and* 5×5 *array. Test your filter on any of the keratocyte images in* `imgs/keratocytes` *Include a screenshot of the resulting mean-filtered images (please include the colorbar in your screenshot).*

Question 2: *Why is it good practice to make the entries in the mean filter sum to 1?*

Question 3: *Play around with the Gaussian and median filters on any of the keratocyte images. What seems to be a good choice for the standard deviation and size for these filters, respectively? Compare the outputs to that of the mean filter and to one another. Include a representative screenshot for each filter.*

These filters, designed to remove noise, are generally referred to as “low-pass” filters, because they smooth out the original signal, leaving the low frequencies while reducing the high frequencies.

What if, instead of denoising the images, we wanted to extract an outline of the cell’s structures? In this case, we want a “high-pass” filter, which will emphasize abrupt changes in intensity rather than overall slow changes.

Question 4: *Implement the* `hp` *and* `gaussianHP` *filters.*
The first filter will convolve the original image with a matrix such as:

$$\begin{bmatrix} -c & -c & -c \\ -c & +8c & -c \\ -c & -c & -c \end{bmatrix}$$

Play around with different values for c and include a representative screenshot.
The second filter exploits the fact that the original image minus a Gaussian low-pass filter gives a reasonable high-pass filter. Include representative screenshots.

Question 5: *Note that in the simple high-pass filter from the previous exercise, the entries in the matrix sum to 0. Why is this a good idea?*

As you can see, the high pass filters are good at picking out the edges of the cellular components, but they are also good at amplifying the noise in the images. Now you will try to improve this extraction process.

Question 6: *Play around with low-pass filtering the output from a high-pass filter and vice-versa. You can use multiple filters on an image by invoking* filter.py *as follows:*

```
python filter.py <path to image file> <filter 1> <filter 2> ...
```

Does alternating or repeating filtering seem to improve the results? Include some screenshots of at least 3 combinations that seem to work well (or that seem to give you garbage), and a brief explanation of why the combinations may behave the way they do.

Now, take a look at the Caulobacter images in *`imgs/caulobacter`*. These images were collected using phase-contrast microscopy, which gives us very good contrast between "Cell" and "Not Cell", but poor contrast within the cell. That said, we can extract the outlines of the cells with a very simple filter.

Question 7: *Implement the* `threshold` *filter. You should select an appropriate value for the threshold and then set all values above this threshold to 0 and all below this threshold to 1. This will invert the image so that the dark cells appear white and the background appears black. Include a representative screenshot and specify the threshold value you used.*

2.2 Principal Component Analysis of Caulobacter Cells

In the next section, we'll use binary masked images, like the ones we generated in the last exercise to analyze cell shapes. In the `masks` directory, there are sequences of binary images that have already been processed so that any part of the image that is a cell is completely white (= 1) and all other parts of the image are completely black (= 0). Let's start by making some qualitative observations.

Question 8: *Describe the shapes of the cells that you see. In particular, what attributes of an individual caulobacter cell's morphology would allow you to distinguish it from the others?*

Now that we have some sense for what the cells in our data set look like, we'll use the software Celltool to help us make these qualitative observations quantitative. Celltool has tools to represent cell shapes as outlines. The first thing we need to do is extract this outline or "contour".

Question 9:
Go in to the `masks` *directory. Extract the contours from* `caulobacter` *as follows.*

```
celltool extract_contours --resample-points=100
--smoothing-factor=0.001 --destination=cauloContours
caulobacter/*.tif
```

Then, run the following command to plot the contours into an `svg` *file.*

```
celltool plot_contours --output-file="cauloContours.svg"
cauloContours/*.contour
```

You should now have a file in the directory called ***cauloContours.svg****. You can view it using a web browser.*
Include a screenshot of `cauloContours.svg` *in your writeup.*

Take a look at `cauloContours.svg`. You should see the outlines of a number of cells laid across the image. It will also be helpful to look at one of the `.contour` files in `cauloContours`. In particular, note that these outlines are saved as an array of (x,y) points. In order to be able to compare and contrast these vectors of points, we must align the vectors such that the ith point in each vector corresponds to an analogous point in each cell.

Question 10:
Align the extracted contours using the following command.

```
celltool align_contours --allow-reflection
--destination=cauloAligned cauloContours/*.contour
```

Then plot these aligned contours.

```
celltool plot_contours --color-by=points
--output-file="cauloAligned.svg" cauloAligned/*.contour
```

Include a screenshot of `cauloAligned.svg` *in your writeup.*

Open `cauloAligned.svg`. Note that the cells should now be aligned, where the indices of points along one cell's outline generally correspond to analogous points on other cells' outlines. Now that we have a consistent representation of the cells, we can begin analyzing the cells quantitatively.

Before talking about our specific data, we should first discuss the main technique that Celltool will use to build a model for the cells: Principal Component Analysis (PCA). PCA is a statistical procedure that identifies a low-dimensional space that can be used to provide a reasonably accurate description of a high-dimensional data set. Let X be our data set of n-dimensional points. Intuitively, we want to find an n-dimensional vector, c, that maximizes the spread between all the points in X when they are projected onto a line of slope c.[1]

[1] This means in theory, all we care about is the slope of c. In practice, the predominant method for calculating the principal components require that you've centered your data X around the origin by subtracting the mean value from each of the data points.

For those who are interested, formally, the first principal component is defined for a data set X centered about the origin as follows.

$$c_1 = \underset{\|c\|=1}{\operatorname{argmax}} \sum_{x \in X} (x \cdot c)^2$$

Recall that a dot product is greatest in absolute value when x and c point along the same axis. Thus, because we assume that X has mean 0 in every coordinate, maximizing the sum of squared dot products results in choosing a direction, c, which maximizes the overall variance after projection.

We'll see that maximizing the variance preserved from the data set allows us to reduce our representation of the data while minimizing the error that is introduced. To get a sense of what this definition means in practice, consider the following example.

Consider the spread of points in Figure **??**. You can view the data more carefully on your own screen by running: `python pca/plotData.py`. Typically, we would describe the coordinates of the points as (x, y) pairs with the bottom left corner at (0,0) moving upward towards the upper right corner at (6, 5).

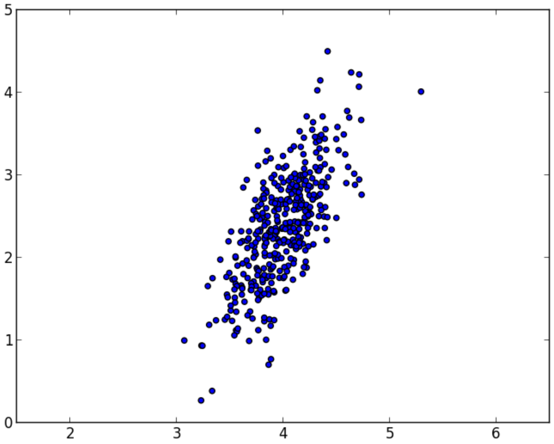

Figure 1: 2D data points

Question 11: *Suppose, to simplify our representation, we wanted to only store one value for each coordinate while maintaining our ability to distinguish between points as much as possible.*

(a) *Clearly, one solution would be to save just the x-coordinate or just the y-coordinate. These solutions project the data onto the lines $y = 0$ and $x = 0$ of slope 0 and ∞, respectively. Use the* `project.py` *script in the* `pca` *directory to see what the data looks like under these projections.*

```
python project.py 0
python project.py inf
```

What is the empirical variance after projection in each case?

(b) *The choice to project onto $x = 0$ and $y = 0$ was arbitrary. By picking the projection line more carefully, we should be able to preserve more of the variance in the data points. Play around with different slope values. What slope maximizes the variance preserved? What is the empirical variance of the data under this projection?*

(c) *The slope you found in the last part is precisely the first principal component — the line that maximizes the spread of points after projection. The second principal component is defined similarly after removing the variation in data points according to the first principal component. In particular, the next principal component must be orthogonal to the first, regardless of the number of dimensions. (You should convince yourself of this fact.) What is the slope and variance preserved of the second principal component?*

Submit a plot of the data projected onto the first principal component and onto the second principal component.

Note that because the data we worked with in the previous question was only 2-dimensional, there are only 2 principal components. In general, if we are dealing with n-dimensional vectors, there will be n principal components. In the case of our cells, where we use 200 values (100 x,y pairs) to represent each cell, there will be 200 principal components, which are each 200-dimensional vectors, which capture all of the variance in the set of cell images. That said, the hope is that the majority of the variance is captured by the first few principal components. If this is the case, then we will have a succinct way of representing and discriminating between individual cells.

Now, we will jump back to using Celltool to investigate the data set of cells.

Question 12:
Go back to the `masks` *directory and run the following commands, which should extract the first two principal components of cells’ shape from the countours you generated before.*

```
celltool shape_model --variance-explained=0.95
--output-prefix="cauloModel" cauloAligned/*.contour
```

Then plot the model.

```
celltool plot_model "cauloModel.contour"
```

Include a screenshot of ***`cauloModel.svg`*** *in your writeup.*

Question 13: *Qualitatively, what do the first two principal components of the Caulobacter data look like? How well do these components correspond to the qualitative descriptions you originally described?*

We can think about these principal components in a few ways. Firstly, this analysis shows that $> 95\%$ of the variance in Caulobacter shape corresponds to only two axes in our original 200-dimensional space. This means, that to a good first approximation, we could represent each cell as 2 numbers, rather than 200 because we only lose 5% of the variance in the population. Additionally, the correspondence between the principal components and our qualitative observations gives a rigorous statistical justification for using these observations as discriminative features. Finally, by using the principal components, we can easily generate new "cells", which we've never seen before, but which are statisically similar to cells that we have observed.

2.3 Characterization of Keratocytes

Now we'd like to use Celltool to characterize cells based on their shape. We've collected images of keratocytes grown in two different media: one in normal media and the other in media diluted to 25% the concentration. Unfortunately, we were not careful in the lab and we mixed up which images correspond to which condition. We also happened to take images of keratocytes grown in another chemical condition that causes a similar osmotic stress on the cells as growing in 25% media. Thus, these cells should look more similar to one another than to the normal cells. We'd like a quantitative method to identify which images correspond to which condition.

Question 14: *Inside the* `keratocytes` *directory the prefix of each image corresponds to the growing conditions, unknowns A and B and known chemical environment X. Build a shape model for the whole data set that will allow you to observe differences in the cell types.*

```
celltool extract_contours --resample-points=100
--smoothing-factor=0.001 --destination=kContours
keratocytes/*/*.tif
```

```
celltool align_contours --allow-reflection
--destination=kAligned kContours/*.contour
```

```
celltool shape_model --variance-explained=0.90
--output-prefix="kModel" kAligned/*.contour
```

```
celltool plot_model "kModel.contour"
```

Include a screenshot of `kModel.svg` *in your writeup.*

Now that we have a model that characterizes the top principal components, we can plot the images as points on the axes of these components. (We'll plot along the first two principal components for ease of visualization.)

Question 15: *Run the following commands to measure the average area of each cell type and degree of each principal component present for each cell type.*

```
celltool measure_contours --output-file="A.csv" --area
--shape-modes kModel.contour 1 2 - kAligned/A*.contour
```

```
celltool measure_contours --output-file="B.csv" --area
--shape-modes kModel.contour 1 2 - kAligned/B*.contour
```

```
celltool measure_contours --output-file="X.csv" --area
--shape-modes kModel.contour 1 2 - kAligned/X*.contour
```

Then plot the distribution of cell types along the axes of the principal components as follows.

```
celltool plot_distribution --x-column=3 --y-column=4
--output-file="dist.svg" A.csv B.csv X.csv kAligned/*.contour
```

Include a screenshot of `dist.svg` *in your writeup. Finally, plot the distribution of areas of the cells as follows.*

```
celltool plot_distribution --x-column=Area
--output-file="areas.svg" A.csv B.csv X.csv
```

Include a screenshot of `areas.svg` *in your writeup.*

Question 16: *Which set of cells was grown in normal media and which was grown in dilute media? How did you come to this conclusion? You should refer to the plots generated in the previous exercise.*

Now that we can effectively analyze and model cellular shape, we can start to talk about the intracellular organization and dynamics.

3 Diffusion Simulations

Once we have a model for the structure of a given type of cell, we frequently want to model the dynamics within the cell. Next, we will look at how biological molecules diffuse throughout cells.

Diffusion is the process by which particles spread out and mix together with other particles. Fundamentally, diffusion is governed by the random motion of individual particles; however, combining many particles' random motions, we can derive deterministic laws that govern the aggregate movement of a substance through solution.

Let's start by thinking about motion in one dimension.

Question 17: *Assume there is a single particle on a number line, starting at position 0, and at each time step it moves left or right by 1 unit with equal probability.*

(a) Give the probability distribution over positions at $t = 4$ and $t = 5$.
Extra Credit: Give a general expression for the probability of the particle being at position $x = x'$ at time $t = t'$.

(b) What is the expected position x at time $t = t'$? What is the expected squared displacement after t' time steps?

Now, let's consider what happens when we add more particles.

Question 18: *Under the same model as the previous question, assume we have 1000 particles, all starting at position 0.*

(a) At the first time step, what is the expected number of particles that pass from $x = 0$ to $x = 1$?

(b) Let $n(x, t)$ be the number of particles at position x at time t. At the t'th time step, what is the expected number of particles that pass from $x = x_0$ to $x = x_0 + 1$? What is the number of particles that pass from $x = x_0 + 1$ to $x = x_0$? Thus, what is the net movement of particles from $x = x_0$ to $x = x_0 + 1$?
Your answers should be expressed in terms of $n(x, t)$.

This result underpins Fick's First Law of Diffusion. The law states that the flux (the net movement of particles across a unit area in a given time interval) is proportional to the concentration gradient.

$$J = -D\frac{\partial C}{\partial x}$$

where J is the flux, C is the concentration, and D is a diffusion coefficient and is given in units of distance^2 per time.

While it is important to understand that particles move from high concentrations to low concentrations in space — and that this fact can be derived from a simple random motion model — what we'd really like is a way to express the change in concentrations over time. If we assume that particles are neither lost nor created, then the change in concentration over time must equal the change in flux over distance.

$$\frac{\partial C}{\partial t} = -\frac{\partial J}{\partial x}$$

Using this fact, we can easily see, by substituting our first expression for J that the change in concentration over time is proportional to the change in concentration *gradient* over space.

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$$

Thus, the macroscopic change in concentration over time is proportional to the degree to which the concentration gradient changes spatially.

Question 19: *Assume that the concentration gradient in a system is constant; that is, $\frac{\partial C}{\partial x} = k$ for some constant k. How does the concentration throughout the system change over time? Explain this macroscopic phenomenon in terms of the individual particles of the system.*

Now, we will put our theoretical understanding to the test and implement two types of diffusion simulations through a cell. We will make the simplifying assumption that cells are rectangles and that if a particle moves beyond a boundary, say to the right, it will appear on the opposite, left-most, boundary. In particular, we will implement one model that updates the number of particles in a given block of the cell stochastically, picking a random direction for each particle, and another that updates the concentration of particles deterministically based on the second derivative of the concentrations (also called the Laplacian).

Question 20: *Implement* `update()` *in* `StochasticDiffuser` *in* `diffuser.py`*.*

Question 21: *Implement* `update()` *in* `LaplacianDiffuser`*.*

Question 22: *Run the* `StochasticDiffuser` *under a variety of initial conditions. Start with the "Single Particle" simulation. How would you characterize the motion of the particle? Next run the "Point Mass" simulation. What does the state of the diffuser look like after 100 iterations? After 500? To what degree does the model "converge"? Can you predict where a specific particle will be after many iterations? Can you predict the distribution of particles after many iterations? Play around with the diffusion coefficient and starting number of particles. Do either of these parameters affect the rate of convergence?*

Question 23: *Run the* `LaplacianDiffuser` *with some of the diffusion conditions in* `diffuser.py`*. Play around with the diffusion coefficient and the starting concentration. Do either of these parameters affect the rate of convergence? Using the "Point Mass" simulation and starting with an initial concentration of $1,250$ and a diffusion coefficient of 0.24, how many iterations does it take for every block to have roughly an equal concentration of particles (i.e. to converge)? Note that our criteria for convergence is: Δconcentration $< \epsilon$ for all voxels, where $\epsilon = 0.001$ (about $1/1,250$).*

Question 24: *As time goes to $+\infty$, what is the probability that the Laplacian model has a block of concentration 0? What about in the stochastic model? Is the probability positive or equal to 0? (Hint: Do NOT try to explicitly compute this probability.)*

Question 25: *Based on the answer to the previous question, and any other observations you've made, when do you think it would be appropriate to use the Laplacian Model versus the stochastic model? What are the benefits and drawbacks to each?*

4 Feedback

Again, we'd love some feedback on the assignment! You will receive full credit for this portion for providing any response. We encourage constructive criticism.

Question 26: *What was your favorite aspect of the assignment? What was your least favorite aspect of the assignment? Why? Any suggestions for improvement?*

Question 27: *Approximately how long did this assignment take you? Where did you spend most of this time?*

5 Submission Instructions

To submit, first go to `https://canvas.stanford.edu/`. Click on CS 279 under the Courses Tab. Next click on the Assignments Tab and select Assignment 3. Save your writeup as `writeup3.pdf`. Upload `writeup3.pdf` and your implementations of `filter.py` and `diffuser.py` to Canvas and submit your assignment.