CS279 Assignment 3

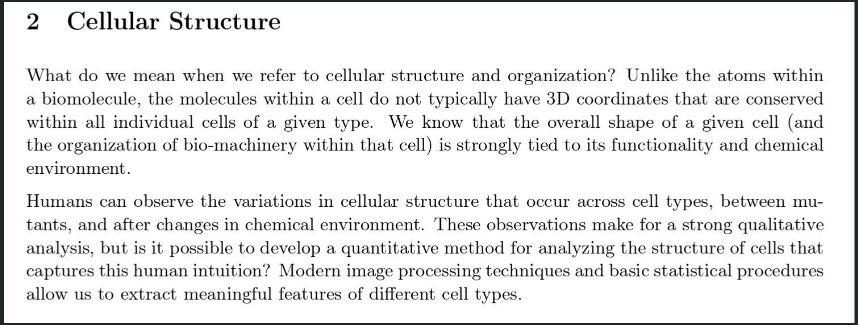
Code Tutorial

Section 2: Cellular Structure

Image Analysis, Principal Component Analysis, and Characterization of Keratocytes

Background Info [lecture 8]

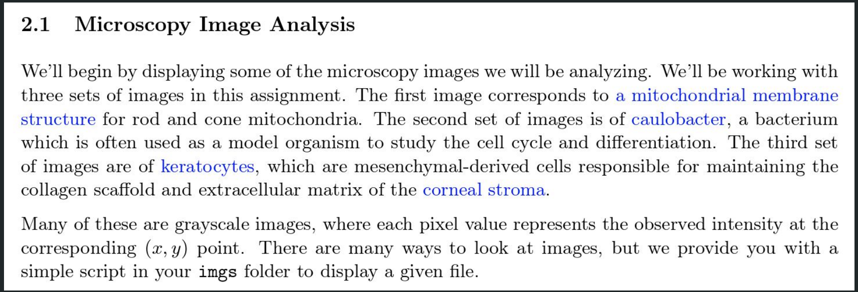
- Imaging helps us learn about the structure of cells
  - Light microscopy, electron microscopy
- Computation is an important part of imaging techniques and processing
- A (grayscale) image can be thought of as:
  - A function of two variables
  - A 2D array of brightness values
  - A matrix of brightness values
- A (color) image can be thought of/treated as:
  - Three separate images, one for each R, G, B
  - A function that returns three values (R, G, B) for each (x, y) coordinate/pair



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.



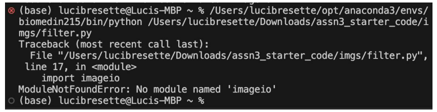
2.1 Microscopy Image Analysis

We'll begin by displaying some of the microscopy images we will be analyzing. We'll be working with three sets of images in this assignment. The first image corresponds to a mitochondrial membrane structure for rod and cone mitochondria. The second set of images is of caulobacter, a bacterium which is often used as a model organism to study the cell cycle and differentiation. The third set of images are of keratocytes, which are mesenchymal-derived cells responsible for maintaining the collagen scaffold and extracellular matrix of the corneal stroma.

Many of these are grayscale images, where each pixel value represents the observed intensity at the corresponding (x, y) point. There are many ways to look at images, but we provide you with a simple script in your `imgs` folder to display a given file.

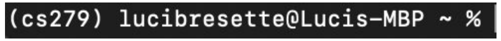
Setup

- LTS machines:
    - If you run your code and get the following error:



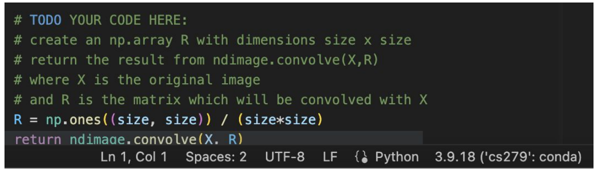
```
(base) lucibresette@Lucis-MBP ~ % /Users/lucibresette/opt/anaconda3/envs/
biomedin215/bin/python /Users/lucibresette/Downloads/assn3_starter_code/i
mgs/filter.py
Traceback (most recent call last):
  File "/Users/lucibresette/Downloads/assn3_starter_code/imgs/filter.py",
 line 17, in <module>
    import imageio
ModuleNotFoundError: No module named 'imageio'
(base) lucibresette@Lucis-MBP ~ %
```

    - Open terminal and use the command: python3.9 -m pip install <module_name>
        - In this case, module_name = imageio
- Local machines: activate your conda environment & install required packages:



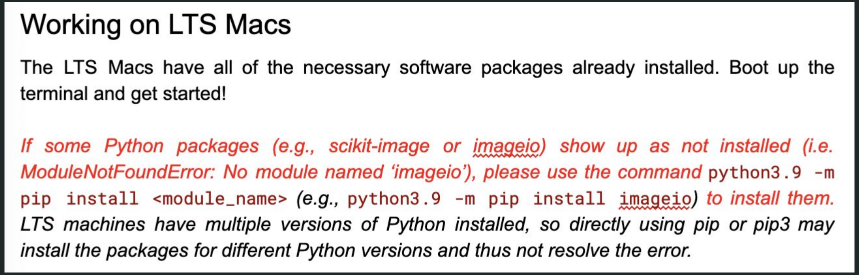
```
(cs279) lucibresette@Lucis-MBP ~ %
```

- Switch your interpreter to the new environment:



```
# TODO YOUR CODE HERE:
# create an np.array R with dimensions size x size
# return the result from ndimage.convolve(X,R)
# where X is the original image
# and R is the matrix which will be convolved with X
R = np.ones((size, size)) / (size*size)
return ndimage.convolve(X, R)
```

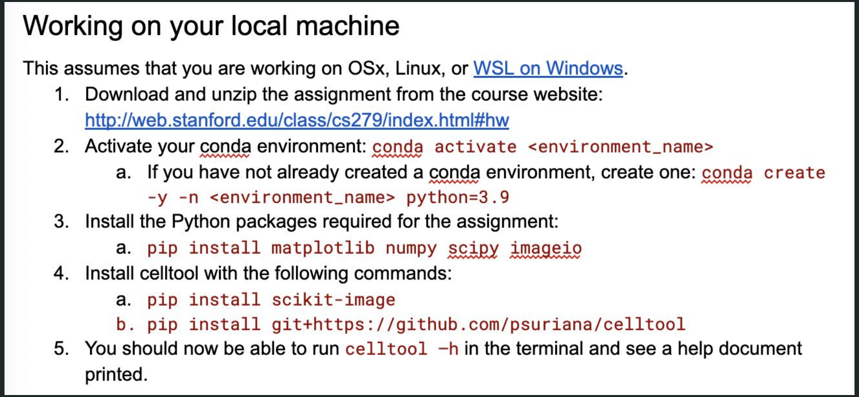
    - Correctness check: celltool -h in terminal prints a help document, and running filter.py does not yield any 'ModuleNotFoundError'



Working on LTS Macs

The LTS Macs have all of the necessary software packages already installed. Boot up the terminal and get started!

If some Python packages (e.g., scikit-image or imageio) show up as not installed (i.e. ModuleNotFoundError: No module named 'imageio'), please use the command `python3.9 -m pip install <module_name>` *(e.g.,* `python3.9 -m pip install imageio`*) to install them. LTS machines have multiple versions of Python installed, so directly using pip or pip3 may install the packages for different Python versions and thus not resolve the error.*



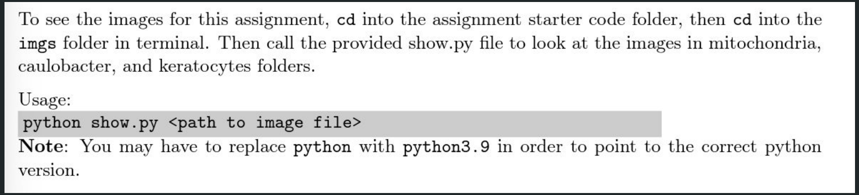
Working on your local machine

This assumes that you are working on OSx, Linux, or WSL on Windows.

1. Download and unzip the assignment from the course website: http://web.stanford.edu/class/cs279/index.html#hw
2. Activate your conda environment: `conda activate <environment_name>`
    a. If you have not already created a conda environment, create one: `conda create -y -n <environment_name> python=3.9`
3. Install the Python packages required for the assignment:
    a. `pip install matplotlib numpy scipy imageio`
4. Install celltool with the following commands:
    a. `pip install scikit-image`
    b. `pip install git+https://github.com/psuriana/celltool`
5. You should now be able to run `celltool -h` in the terminal and see a help document printed.

Setup

- To copy pathname:
  - Mac: right-click on file ➜ hold option key ➜ ‘copy [filename] as Pathname’
  - Press and hold the shift key ➜ right-click on folder ➜ “Copy as path”
- Once in imgs folder:
  - Two approaches: use [imgFolder]/[imgFileName] or ‘copy pathname’ each time
- You won’t be able to put more commands in the terminal until you exit the image
- LTS users: make sure to use ‘python3.9’ (no space)

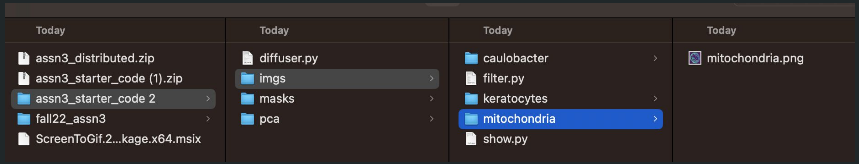


To see the images for this assignment, `cd` into the assignment starter code folder, then `cd` into the `imgs` folder in terminal. Then call the provided show.py file to look at the images in mitochondria, caulobacter, and keratocytes folders.

Usage:

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

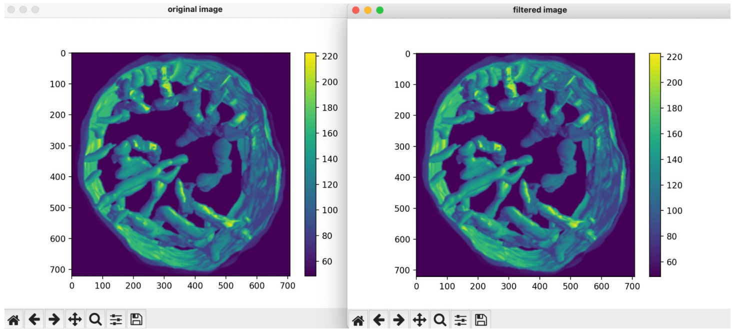
Note: You may have to replace `python` with `python3.9` in order to point to the correct python version.

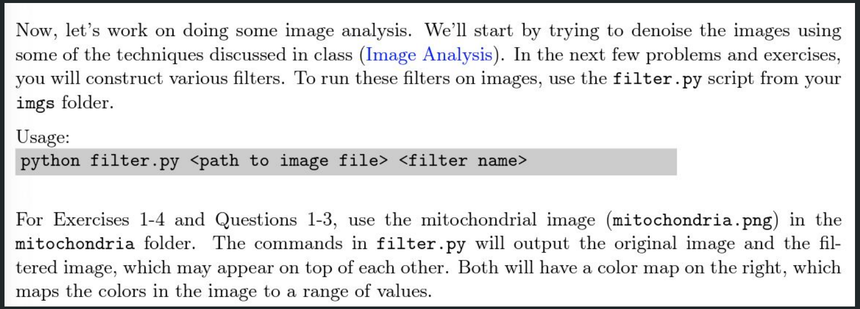


Today	Today	Today	Today
assn3_distributed.zip	diffuser.py	caulobacter	mitochondria.png
assn3_starter_code (1).zip	imgs	filter.py	
assn3_starter_code 2	masks	keratocytes	
fall22_assn3	pca	mitochondria	
ScreenToGif.2...kage.x64.msix		show.py	

Setup

- Both the original and filtered images are output, but they might be on top of each other
- The color map on the right maps the colors in the image to a range of values; hover over a pixel for precise values at that location

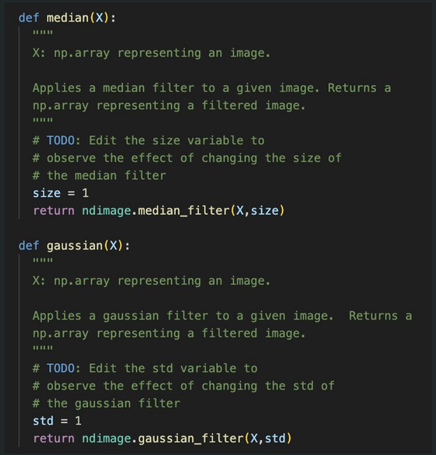



Now, let's work on doing some image analysis. We'll start by trying to denoise the images using some of the techniques discussed in class (Image Analysis). In the next few problems and exercises, you will construct various filters. To run these filters on images, use the `filter.py` script from your `imgs` folder.

Usage:

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

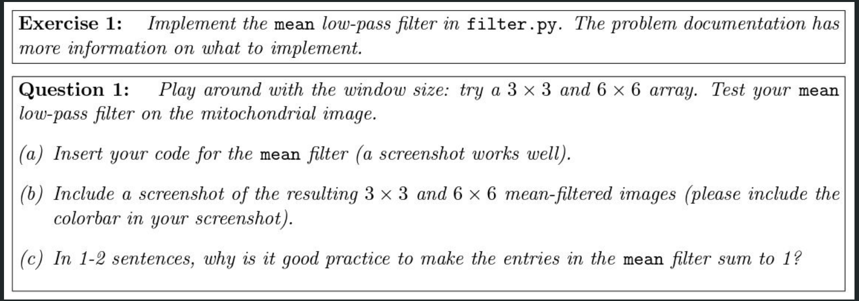
For Exercises 1-4 and Questions 1-3, use the mitochondrial image (`mitochondria.png`) in the `mitochondria` folder. The commands in `filter.py` will output the original image and the filtered image, which may appear on top of each other. Both will have a color map on the right, which maps the colors in the image to a range of values.



```python
def median(X):
    """
    X: np.array representing an image.

    Applies a median filter to a given image. Returns a
    np.array representing a filtered image.
    """
    # TODO: Edit the size variable to
    # observe the effect of changing the size of
    # the median filter
    size = 1
    return ndimage.median_filter(X,size)

def gaussian(X):
    """
    X: np.array representing an image.

    Applies a gaussian filter to a given image.  Returns a
    np.array representing a filtered image.
    """
    # TODO: Edit the std variable to
    # observe the effect of changing the std of
    # the gaussian filter
    std = 1
    return ndimage.gaussian_filter(X,std)
```

Exercise 1

- Low-pass filter: smooths an image by reducing its high-frequency components
- ‘Mean filter’: replaces each pixel value with the average of its neighbors, including itself
- An image can be represented as a 2D array where each element in the array is a pixel value
  - In the starter code, this is ‘X’
- Create an np.array R: np.ones((num, num))
- The size of the matrix defines the ‘neighborhood’ over which we’ll be taking the average
  - Size = 1: consider just the pixel
  - Size = 3: consider immediate neighbors
- Convolution is a mathematical operation that combines two arrays (in this case, an image and a filter) to produce a third array (the filtered image)
  - In image processing, convolution is like applying a sliding window over the image and calculating a weighted sum at each position
  - Imagine the filter as a smaller window we slide over the image. At each position, the filter multiplies each corresponding pair of filter & image pixels, then sums all these multiplied values
  - This sum then becomes a single pixel value in the output image
- **What type of filter, regardless of size, will take the average of exactly the pixels it is placed over?**



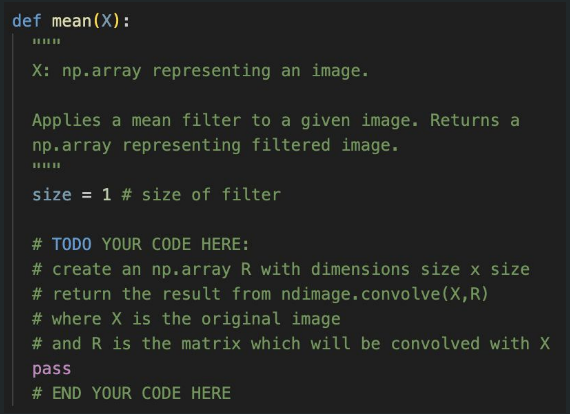
Exercise 1: *Implement the* `mean` *low-pass filter in* `filter.py`*. The problem documentation has more information on what to implement.*

Question 1: *Play around with the window size: try a 3×3 and 6×6 array. Test your* `mean` *low-pass filter on the mitochondrial image.*

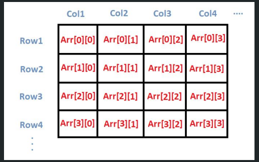
(a) Insert your code for the `mean` *filter (a screenshot works well).*

(b) Include a screenshot of the resulting 3×3 and 6×6 mean-filtered images (please include the colorbar in your screenshot).

(c) In 1-2 sentences, why is it good practice to make the entries in the `mean` *filter sum to 1?*



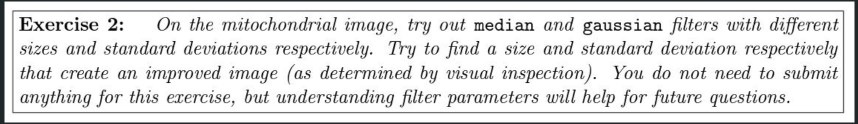
```python
def mean(X):
    """
    X: np.array representing an image.

    Applies a mean filter to a given image. Returns a
    np.array representing filtered image.
    """
    size = 1 # size of filter

    # TODO YOUR CODE HERE:
    # create an np.array R with dimensions size x size
    # return the result from ndimage.convolve(X,R)
    # where X is the original image
    # and R is the matrix which will be convolved with X
    pass
    # END YOUR CODE HERE
```



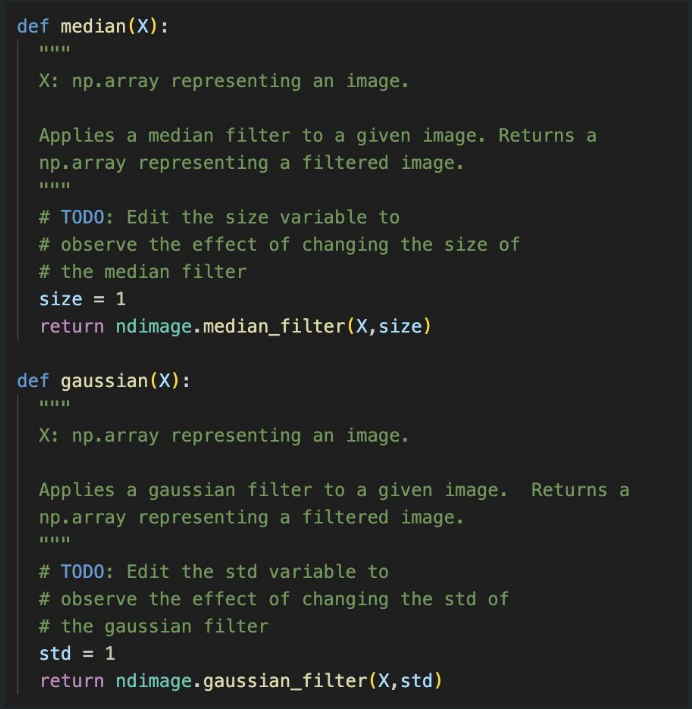
	Col1	Col2	Col3	Col4	....
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]	
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]	
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]	
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]	
⋮					

Exercise 2

- Edit the numbers
- Run the code
- Evaluate the images with the new numbers:
  - python filter.py mitochondria/mitochondria.png median
  - python filter.py mitochondria/mitochondria.png gaussian
- Recommendation: create a separate doc tracking the different numbers with screenshots of the outputs to refer back to later



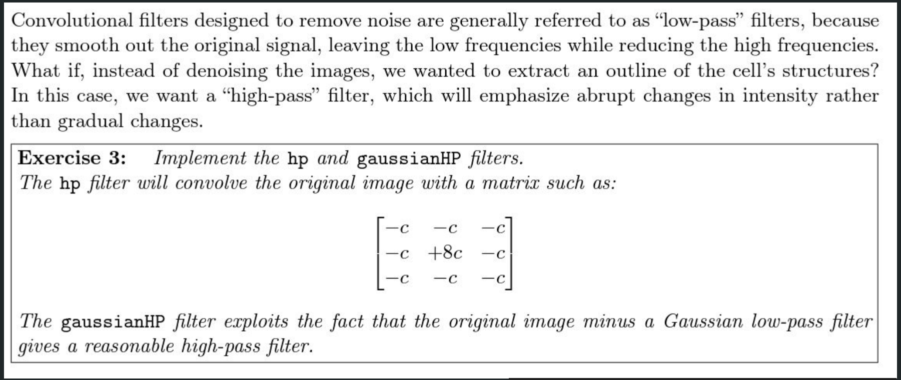
Exercise 2: *On the mitochondrial image, try out* `median` *and* `gaussian` *filters with different sizes and standard deviations respectively. Try to find a size and standard deviation respectively that create an improved image (as determined by visual inspection). You do not need to submit anything for this exercise, but understanding filter parameters will help for future questions.*



```python
def median(X):
    """
    X: np.array representing an image.

    Applies a median filter to a given image. Returns a
    np.array representing a filtered image.
    """
    # TODO: Edit the size variable to
    # observe the effect of changing the size of
    # the median filter
    size = 1
    return ndimage.median_filter(X,size)

def gaussian(X):
    """
    X: np.array representing an image.

    Applies a gaussian filter to a given image.  Returns a
    np.array representing a filtered image.
    """
    # TODO: Edit the std variable to
    # observe the effect of changing the std of
    # the gaussian filter
    std = 1
    return ndimage.gaussian_filter(X,std)
```

Exercise 3

- hp filter:
    - Using the example matrix subtracts the average of the surrounding pixels from the center pixel ➜ more contrast, and highlights regions of rapid intensity change
    - The higher the c_var, the greater difference between the central pixel and the surrounding pixels
    - *Understanding check: why is the center factor in the example matrix 8 while the surrounding factors are -1?*
- gaussianHP filter:
    - If you remove the low-frequency components of an image, you're left with the high-frequency components

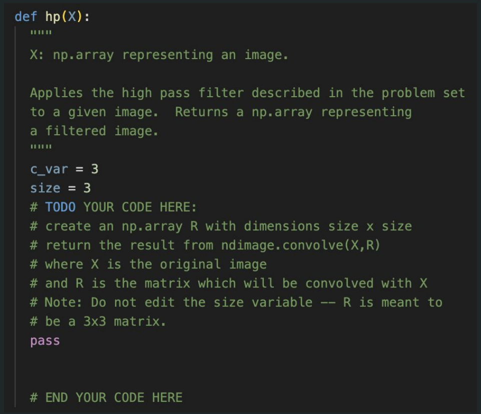


Convolutional filters designed to remove noise are generally referred to as "low-pass" filters, because they smooth out the original signal, leaving the low frequencies while reducing the high frequencies. What if, instead of denoising the images, we wanted to extract an outline of the cell's structures? In this case, we want a "high-pass" filter, which will emphasize abrupt changes in intensity rather than gradual changes.

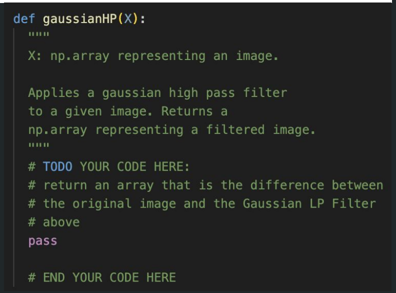
Exercise 3: *Implement the* `hp` *and* `gaussianHP` *filters.*
The `hp` *filter will convolve the original image with a matrix such as:*

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

The `gaussianHP` *filter exploits the fact that the original image minus a Gaussian low-pass filter gives a reasonable high-pass filter.*



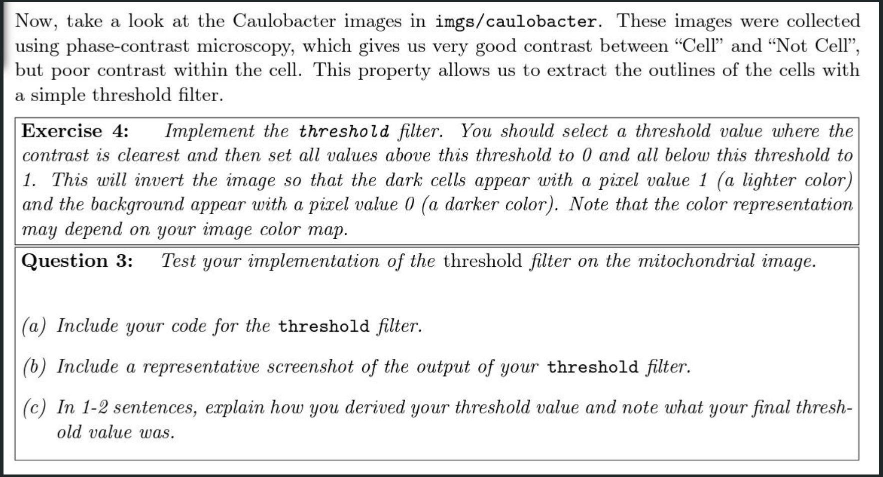
```python
def hp(X):
    """
    X: np.array representing an image.

    Applies the high pass filter described in the problem set
    to a given image.  Returns a np.array representing
    a filtered image.
    """
    c_var = 3
    size = 3
    # TODO YOUR CODE HERE:
    # create an np.array R with dimensions size x size
    # return the result from ndimage.convolve(X,R)
    # where X is the original image
    # and R is the matrix which will be convolved with X
    # Note: Do not edit the size variable -- R is meant to
    # be a 3x3 matrix.
    pass


    # END YOUR CODE HERE
```



```python
def gaussianHP(X):
    """
    X: np.array representing an image.

    Applies a gaussian high pass filter
    to a given image. Returns a
    np.array representing a filtered image.
    """
    # TODO YOUR CODE HERE:
    # return an array that is the difference between
    # the original image and the Gaussian LP Filter
    # above
    pass

    # END YOUR CODE HERE
```

Exercise 4

- python show.py caulobacter/WT.tif
  - Get a sense of the pixel values by exploring this image (hover over the pixels to get coordinates & value)
- Element-wise comparison in NumPy arrays:
  - someVal = 5
  - Arr = [0, -1, 4, 7, 8, 0]
  - threshArr = (Arr < 5)
  - #threshArr = [False, False, False, True, True, False]
  - **How to go from boolean values to 0s and 1s?**
- Some of this exercise will depend on trial and error



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. This property allows us to extract the outlines of the cells with a simple threshold filter.

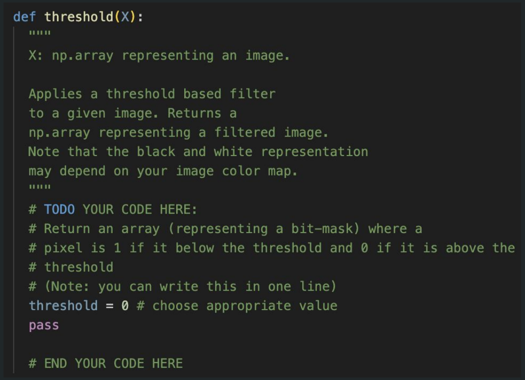
Exercise 4: *Implement the* ***`threshold`*** *filter. You should select a threshold value where the contrast is clearest and then set all values above this threshold to 0 and all below this threshold to 1. This will invert the image so that the dark cells appear with a pixel value 1 (a lighter color) and the background appear with a pixel value 0 (a darker color). Note that the color representation may depend on your image color map.*

Question 3: *Test your implementation of the* threshold *filter on the mitochondrial image.*

(a) *Include your code for the* `threshold` *filter.*

(b) *Include a representative screenshot of the output of your* `threshold` *filter.*

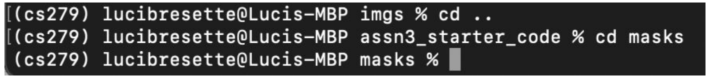
(c) *In 1-2 sentences, explain how you derived your threshold value and note what your final threshold value was.*



```python
def threshold(X):
    """
    X: np.array representing an image.

    Applies a threshold based filter
    to a given image. Returns a
    np.array representing a filtered image.
    Note that the black and white representation
    may depend on your image color map.
    """
    # TODO YOUR CODE HERE:
    # Return an array (representing a bit-mask) where a
    # pixel is 1 if it below the threshold and 0 if it is above the
    # threshold
    # (Note: you can write this in one line)
    threshold = 0 # choose appropriate value
    pass

    # END YOUR CODE HERE
```

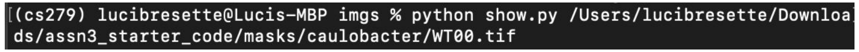
Questions 4 & 5

- [if currently in imgs]:
  - cd ..
  - cd masks

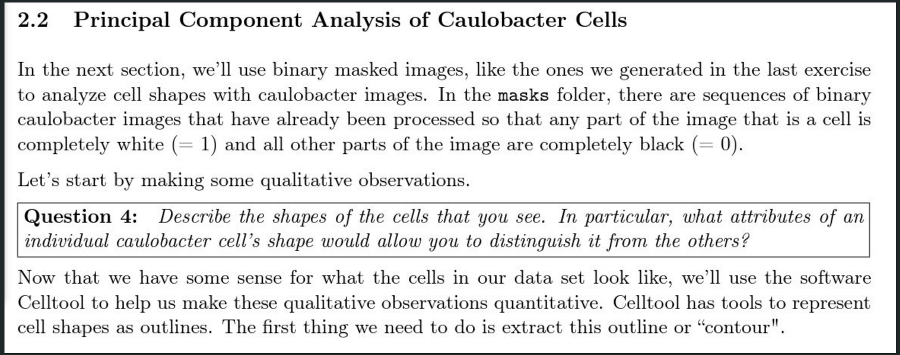


```
[(cs279) lucibresette@Lucis-MBP imgs % cd ..
[(cs279) lucibresette@Lucis-MBP assn3_starter_code % cd masks
(cs279) lucibresette@Lucis-MBP masks % 
```

- Python show.py caulobacter/WT00.tif
  - WT00 to WT18
- To use show.py: use full pathname
  - Otherwise, can view images as normal



```
[(cs279) lucibresette@Lucis-MBP imgs % python show.py /Users/lucibresette/Downloa
ds/assn3_starter_code/masks/caulobacter/WT00.tif
```



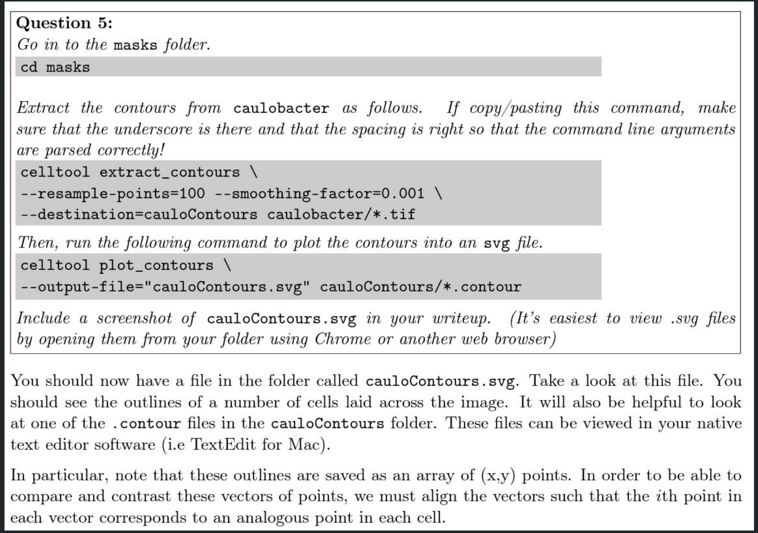
2.2 Principal Component Analysis of Caulobacter Cells

In the next section, we'll use binary masked images, like the ones we generated in the last exercise to analyze cell shapes with caulobacter images. In the `masks` folder, there are sequences of binary caulobacter 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 4: *Describe the shapes of the cells that you see. In particular, what attributes of an individual caulobacter cell's shape 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 5:
Go in to the `masks` *folder.*

```
cd masks
```

Extract the contours from `caulobacter` *as follows. If copy/pasting this command, make sure that the underscore is there and that the spacing is right so that the command line arguments are parsed correctly!*

```
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
```

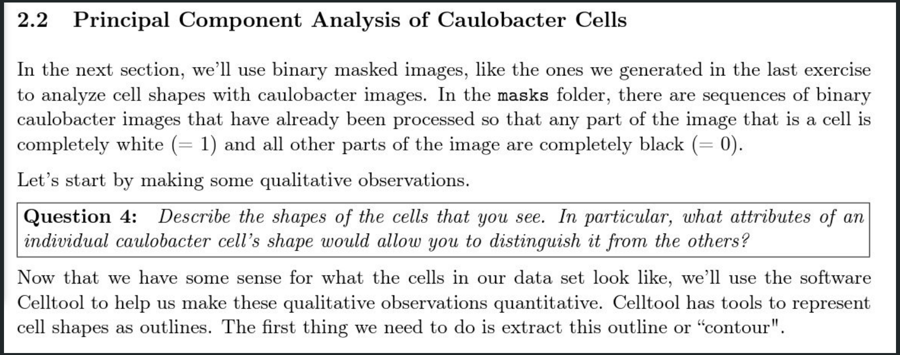
Include a screenshot of `cauloContours.svg` *in your writeup. (It's easiest to view .svg files by opening them from your folder using Chrome or another web browser)*

You should now have a file in the folder called `cauloContours.svg`. Take a look at this file. 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 the `cauloContours` folder. These files can be viewed in your native text editor software (i.e TextEdit for Mac).

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.

Questions 4 & 5

- Full pathname: /Users/lucibresette/Downloads/assn3_starter_code/masks
- Recommendation: type command rather than copy/paste
    - Alternatively, copy into a separate document/app, edit, then copy into terminal
    - The \ character is a newline and does not need to be copied
- Functionality checks:
    - After first command: cauloContours folder created
    - After second command: masks contains cauloContours.svg
        - Open with → google chrome
- To open a .contour file: select any file in the cauloContours folder → Open with → TextEdit (or any other text editor)



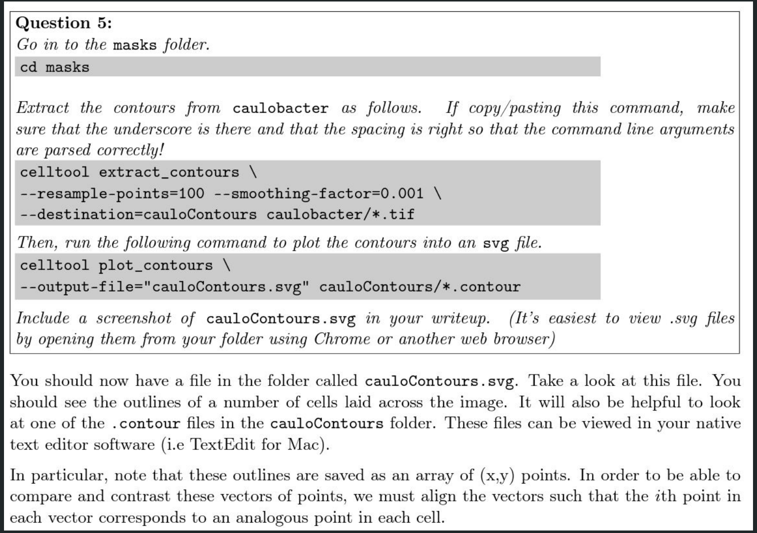
2.2 Principal Component Analysis of Caulobacter Cells

In the next section, we'll use binary masked images, like the ones we generated in the last exercise to analyze cell shapes with caulobacter images. In the **masks** folder, there are sequences of binary caulobacter 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 4: *Describe the shapes of the cells that you see. In particular, what attributes of an individual caulobacter cell's shape 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 5:
Go in to the `masks` *folder.*

```
cd masks
```

Extract the contours from `caulobacter` *as follows. If copy/pasting this command, make sure that the underscore is there and that the spacing is right so that the command line arguments are parsed correctly!*

```
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
```

Include a screenshot of `cauloContours.svg` *in your writeup. (It's easiest to view .svg files by opening them from your folder using Chrome or another web browser)*

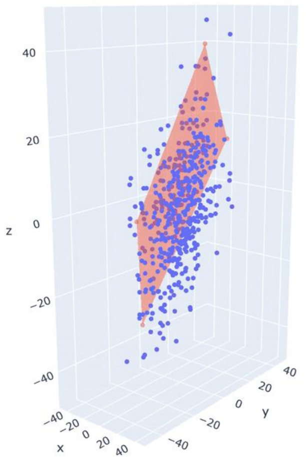
You should now have a file in the folder called `cauloContours.svg`. Take a look at this file. You should see the outlines of a number of cells laid across the image. It will also be helpful to look at one of the `.contour` files in the `cauloContours` folder. These files can be viewed in your native text editor software (i.e TextEdit for Mac).

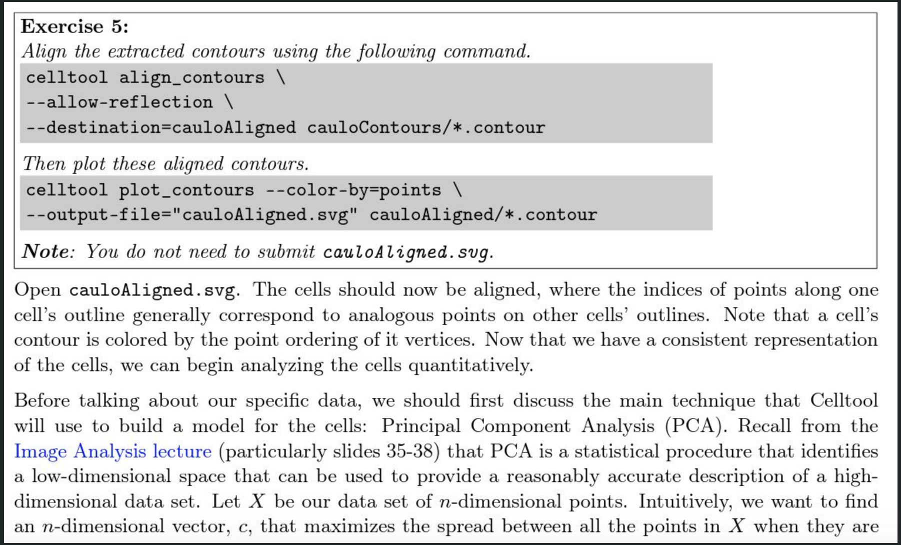
In particular, note that these outlines are saved as an array of (x,y) points. In order to be able to compare and contrast these vectors of points, we must align the vectors such that the ith point in each vector corresponds to an analogous point in each cell.

Exercise 5

- Functionality check:
  - After first section: folder 'cauloAligned' created in 'masks'
  - After second section: 'cauloAligned.svg' created in 'masks'
- Principal component analysis:

Essentially, identifies a lower-dimensional space to describe a high-dimensional dataset





Exercise 5:
Align the extracted contours using the following command.

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

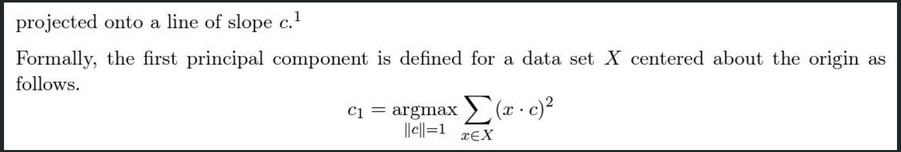
Then plot these aligned contours.

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

***Note**: You do not need to submit `cauloAligned.svg`.*

Open `cauloAligned.svg`. 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. Note that a cell's contour is colored by the point ordering of it vertices. 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). Recall from the Image Analysis lecture (particularly slides 35-38) that 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]

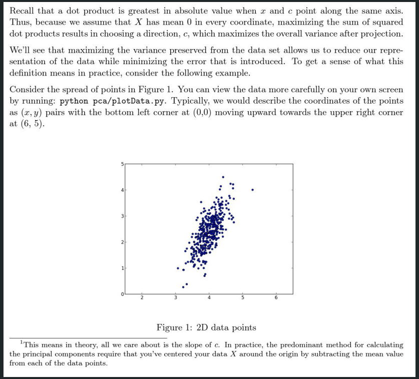
Formally, the first principal component is defined for a data set X centered about the origin as follows.

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

I.e., find a straight line (defined by the vector c) such that when all the points are 'projected' or 'dropped' on this line, they are spread out as much as possible

Question 6

- Have to cd out of masks before running python pca/plotData.py:
  - [from masks directory] cd ..
  - Or, use full pathname: python /Users/lucibresette/Downloads/assn3_starter_code/pca/plotData.py
- Then:
  - cd ..
  - Should be in assignment folder directory
  - Can then run python project.py 0, python project.py inf



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 1. 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).

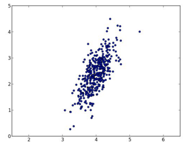
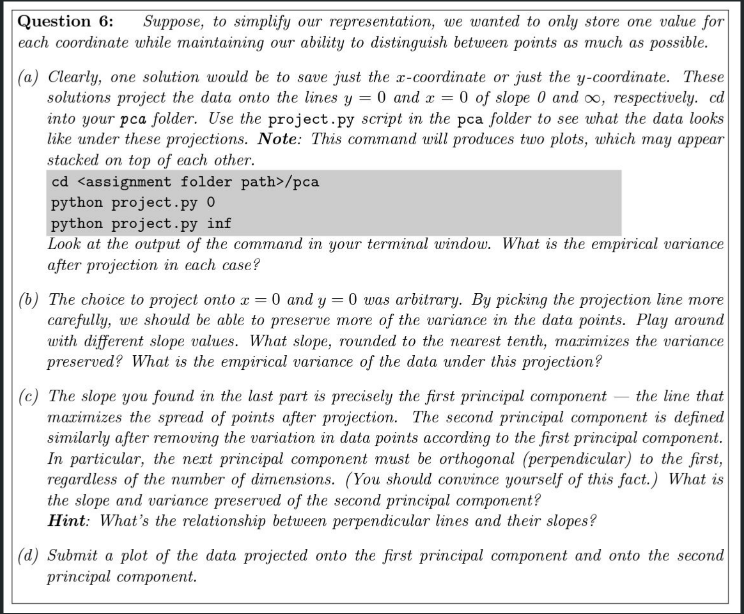


Figure 1: 2D data points

[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.



Question 6: *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. cd into your* `pca` *folder. Use the* `project.py` *script in the* `pca` *folder to see what the data looks like under these projections.* ***Note****: This command will produces two plots, which may appear stacked on top of each other.*

```
cd <assignment folder path>/pca
python project.py 0
python project.py inf
```

Look at the output of the command in your terminal window. What is the empirical variance after projection in each case?

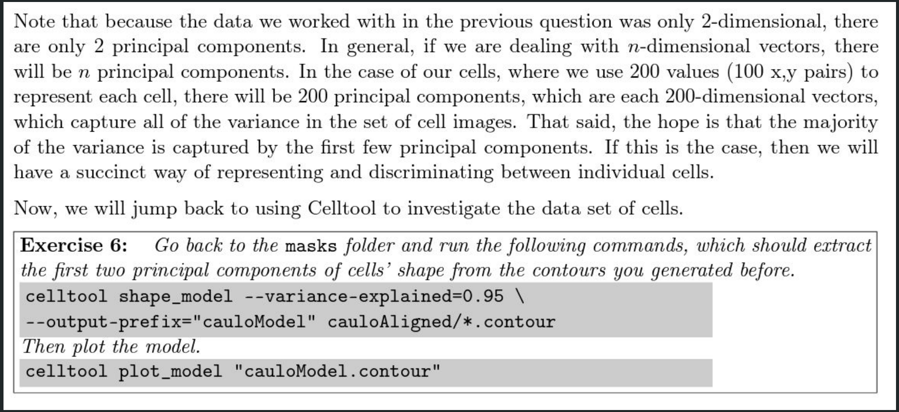
(b) *The choice to project onto $x = 0$ and $y = 0$ was arbitrary. By picking the projection line more carefully, we should be able to preserve more of the variance in the data points. Play around with different slope values. What slope, rounded to the nearest tenth, maximizes the variance preserved? 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 (perpendicular) 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?*
Hint*: What's the relationship between perpendicular lines and their slopes?*

(d) *Submit a plot of the data projected onto the first principal component and onto the second principal component.*

Exercise 6

- Go back into masks folder:
  - If currently in pca folder:
    - cd ..
    - cd masks
  - Or, use full pathname: cd /Users/lucibresette/Downloads/assn3_starter_code/masks
- Functionality checks:
  - For the first part: file ‘cauloModel.contour’ created in masks folder
  - Second part: cauloModel.svg created in masks folder



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.

Exercise 6: *Go back to the* `masks` *folder and run the following commands, which should extract the first two principal components of cells' shape from the contours you generated before.*

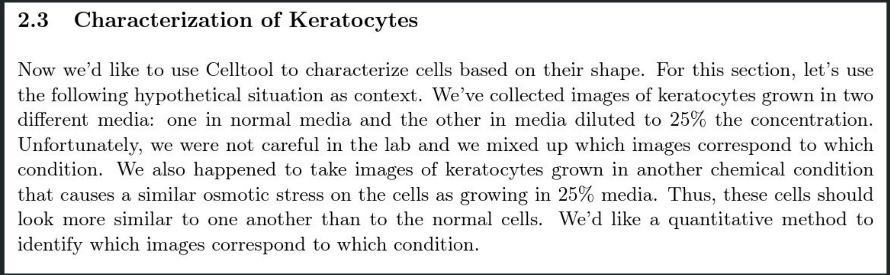
```
celltool shape_model --variance-explained=0.95 \
--output-prefix="cauloModel" cauloAligned/*.contour
```

Then plot the model.

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

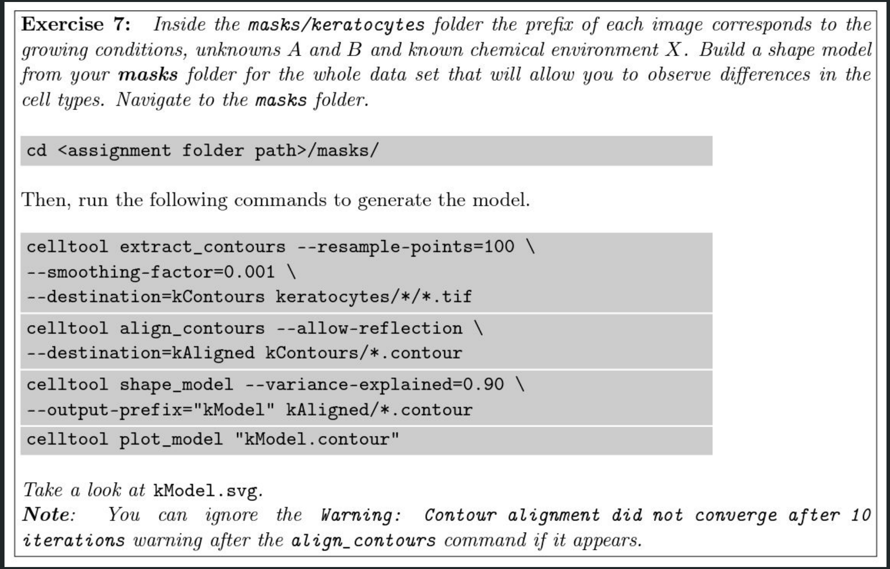
Exercise 7

- Navigate to the masks folder:
  - As full path: /Users/lucibresette/Downloads/assn3_starter_code/masks
- Functionality checks:
  - After first block: folder ‘kContours’ created inside ‘masks’
  - After second block: folder ‘kAligned’ created inside ‘masks’
  - After third block: files ‘kModel-normalized-positions.csv’, ‘kModel-positions.csv’, and ‘kModel.contour’ created in ‘masks
  - After fourth block: ‘kModel.svg” created inside ‘masks’



2.3 Characterization of Keratocytes

Now we’d like to use Celltool to characterize cells based on their shape. For this section, let’s use the following hypothetical situation as context. 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.



Exercise 7: *Inside the* ***masks/keratocytes*** *folder the prefix of each image corresponds to the growing conditions, unknowns A and B and known chemical environment X. Build a shape model from your* ***masks*** *folder for the whole data set that will allow you to observe differences in the cell types. Navigate to the* ***masks*** *folder.*

```
cd <assignment folder path>/masks/
```

Then, run the following commands to generate the model.

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

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

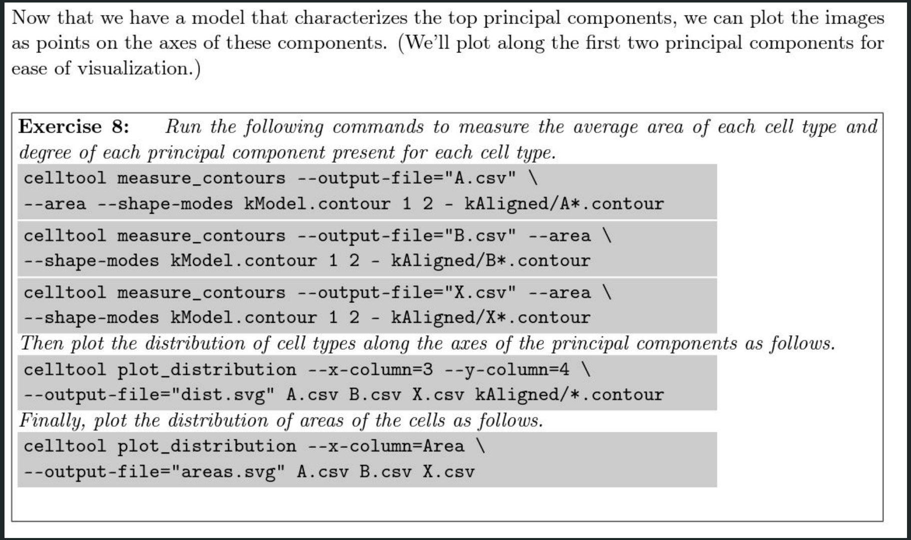
```
celltool shape_model --variance-explained=0.90 \
--output-prefix="kModel" kAligned/*.contour
```

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

Take a look at `kModel.svg`.
Note: *You can ignore the* ***Warning: Contour alignment did not converge after 10 iterations*** *warning after the* ***align_contours*** *command if it appears.*

Exercise 8

- Functionality checks (note these are all in the ‘masks’ folder):
  - After first block: file ‘A.csv’ created
  - After second block: file ‘B.csv’ created
  - After third block: file ‘X.csv’ created
  - After fourth block: file ‘dist.svg’ created
  - After fifth block: file ‘areas.svg’ created



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.)

Exercise 8: *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
```

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
```