Assignment 3 – Cellular Structure and Dynamics

CS/BIOE/CME/BIOPHYS/BIOMEDIN 279

Due: Nov 17, 2022 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. 140–156 (2007).

1 Preliminaries

- **LTS** users: Please make sure to use the command "python3.9" instead of "python" for every block of code where "python" is currently written.
- For this assignment, we **strongly recommend** using LTS machines for the coding portion of the assignment. Installing packages directly on Windows can be challenging, and we may be unable to provide support with this process on Windows. Mac, WSL on Windows, and Linux users have a better chance of success with self-installation, but it has proven difficult for many students in the past. See the Assignment 3 Setup Document for details on software installation and/or setup.
- We request that students living on campus utilize in-person LTS clusters. Students who live off campus and are not on campus frequently can use remotely accessible LTS machines. The supply of remotely accessible LTS machines is limited, so you may not be able to access one just before the assignment deadline; please plan accordingly.
- Please start the assignment early so that we have time to address any software challenges in office hours and via Ed well before the deadline.
- We'll ask you to submit screenshots of a few functions throughout the assignment. Please only submit the functions we request. We do not want any other code submitted.

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.

To see the images for this assignment, cd into your assn3/imgs/ folder in terminal. Then call the provided show.py file to look at the images in mitochondria, caulobacter, and keratocytes.

Usage:

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

Now, let's work on dong some image analysis. We'll start by trying to denoise the images using some of the techniques discussed in class. 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 1-3 and Questions 1-2, use the mitochondrial image.

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?*

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.*

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 overall slow changes.

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

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

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

Question 2: *Play around with different values for* c.

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

(b) Include a representative screenshot of the output of your `hp` *filter, using the final value for* c *you choose.*

(c) Include your code for the `gaussianHP` *filter.*

(d) Include a representative screenshot of the output of your `gaussianHP` *filter.*

(e) Note that in the simple `hp` *filter, the entries in the matrix sum to 0. Why is this a good idea?*

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 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 with a pixel value 1 (white) and the background appear with a pixel value 0 (black). Note that the black and white 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.

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` directory, 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 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 5:
Go in to the `masks` *directory.*

```
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 directory called `cauloContours.svg`. Take a look at this file. (The best way to view is probably in a web browser.) 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.

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
```

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]

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 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).

[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.

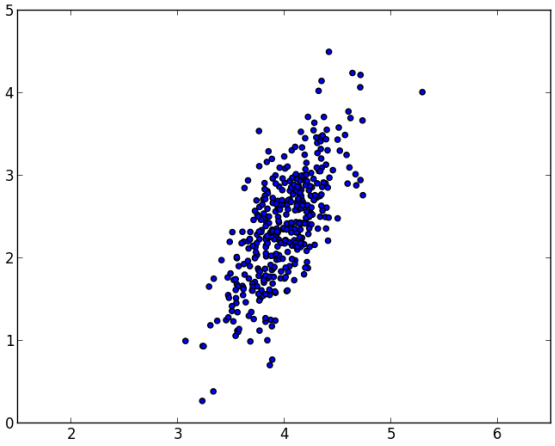

Figure 1: 2D data points

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*** *directory. Use the* `project.py` *script in the* `pca` *directory to see what the data looks like under these projections.*

```
cd <assignment folder path>/pca
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?

(d) 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.

Exercise 6: *Go back to the* `masks` *directory 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"
```

Question 7: *Qualitatively, what do the first two principal components of the Caulobacter data look like? What differences are they capturing? How well do these components correspond to the qualitative descriptions you originally described (question 4)? Include a screenshot of* `cauloModel.svg` *in your writeup.*

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 images of 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. For this section, let's use this 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` *directory 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*** *directory for the whole data set that will allow you to observe differences in the cell types.*

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

```
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`.

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
```

Question 8: *Take a look at your outputs from Exercise 8. We'll use these files to answer Question 9.*

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

Question 9: *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 intra-cellular 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 10: *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 $t = 5$. In other words, list the probability of the particle being at each possible position, at $t = 4$ and $t = 5$ respectively. A two-column table (one column of position, one column of probability) has historically worked well.
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 11: *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. After t' time steps, what is the expected number of particles that pass from $x = x_0$ to $x = x_0 + 1$? What is the expected number of particles that pass from $x = x_0 + 1$ to $x = x_0$? And combining these two numbers, 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 Law," which states that the flux (the net movement of particles across

a unit area per unit time) 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 and substituting our first expression for J, we can derive the diffusion equation, which states 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, at a given point in space, the change in local concentration over time is proportional to the degree to which the concentration gradient changes spatially.

Question 12: *Assume that concentration changes linearly across a system, such that the concentration gradient is constant: $\frac{\partial C}{\partial x} = k$ for some constant k. How does the concentration throughout the system change over time? Explain this macroscopic phenomenon in terms of the motions of individual particles of the system.*

Now, we will put our theoretical understanding to the test and implement two types of simulations of particles diffusing through a cell. We will make the simplifying assumption that cells are rectangles and that if a particle moves beyond a boundary, say over the rightmost boundary, it will appear on the opposite, leftmost, boundary. We will have you complete the code necessary to update the state of the cell from one time step to the next.

In the `StochasticDiffuser`, you will fill in the number of particles at each block (cell) for the next state (`newBlocks`) by iterating through the current cell state and randomly assigning a new direction for each particle to move.

In the `LaplacianDiffuser`, you will update the "concentration" of particles deterministically based on the discrete second derivative of the concentrations (also called the Laplacian).

Before you get started, carefully read over the code in `diffuser.py`. You should understand what `self.blocks` represents and where the initial simulation parameters are set, as well as how you would use the functions `self.nextX()` and `self.nextY()`.

Exercise 9: *Implement* `update()` *in* `StochasticDiffuser` *in* `diffuser.py`.

Exercise 10: *Implement* `update()` *in* `LaplacianDiffuser` *in* `diffuser.py`.

Hint: To derive the Laplacian, we will approximate the appropriate derivatives using finite differences (i.e. $\frac{dF}{dx} \approx \frac{f(x+h)-f(x)}{h}$*). Specifically, you will approximate the second derivative of concentration with respect to x (*$\frac{d^2C}{dx^2}$*) or y (*$\frac{d^2C}{dy^2}$*). This means that you will also approximate the first derivatives,* $\frac{dC}{dx}$ *and* $\frac{dC}{dy}$*. Assume that the values stored in self.blocks represent concentrations.*

These approximations can be numerically unstable if both the first and second derivatives are calculated using the same interval, i.e. $\frac{df}{dx} = \frac{f(x+h)-f(x)}{h}$ *and* $\frac{d^2f}{dx^2} = \frac{df_x(x+h)-df_x(x)}{h}$ *are using the same interval between* x *and* $x+h$*. To avoid instability, you will need to balance the derivatives so that they are calculated using different intervals, i.e.* $\frac{df}{dx} = \frac{f(x+h)-f(x)}{h}$ *and* $\frac{d^2f}{dx^2} = \frac{df_x(x)-df_x(x-h)}{h}$*.*

Note: If you need to stop the simulations below, you may do so in terminal via control-C.

Question 13: *Run the* `StochasticDiffuser` *(*`python3.9 diffuser.py stoc`*) under a variety of initial conditions. You can locate pre-populated conditions at the end of* `diffuser.py`*, and un-comment the condition to run.*

(a) Include your code for `update` *in* `StochasticDiffuser`*.*
You may remove the comments inside the function if it takes up too much space.

(b) Start with the "Single Particle" simulation. How would you characterize the motion of the particle?

(c) 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"?
Hint: You can edit the plotting script to make it only plot once after N iterations. You may include screenshots if it helps your explanation, but this is not necessary.

(d) Can you predict where a specific particle will be after many iterations? Can you predict the distribution of particles after many iterations?

(e) Play around with the diffusion coefficient and starting number of particles. Do either of these parameters affect the rate of convergence?

Question 14: *Run the* `LaplacianDiffuser` *(*`python3.9 diffuser.py lap`*) with some of the diffusion conditions in* `diffuser.py`*.*

(a) Include your code for `update` *in* `LaplacianDiffuser`*.*
You may remove the comments inside the function if it takes up too much space.

(b) Play around with the diffusion coefficient and the starting concentration. Do either of these parameters affect the rate of convergence?

(c) 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:* $\Delta concentration < (\epsilon * initial\ concentration)$ *for all voxels, where* $\epsilon = 0.001$ *(about* $1/1,250$*). Provide a brief description of how you determined when the simulation converged. (Hint: you can do an approximate calculation using the scale bar to estimate the number of iterations needed for convergence.)*

Question 15: *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. We are asking you to use your intuition based on how these two models work.)*

Question 16: *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 17: *What was your favorite aspect of the assignment? What was your least favorite aspect of the assignment? Why? Any suggestions for improvement?*

Question 18: *Approximately how long did this assignment take you? Where did you spend most of this time?*

5 Challenge Problem

Instructions for the challenge problem can be found in this document. The challenge problem dataset is uploaded to the website under the assignments section.

Include the following in your submission:

- A binary segmented image of each of the four input files, e.g. nuclei white, background black.

- Your histograms of areas and intensities. Your scatter plot of area versus intensity.
- A complete explanation of your analysis pipeline. Illustrate each step of your pipeline with example images. Why did you design it this way?
- An explanation of other approaches you tried. Why did they perform more poorly? Illustrate with example images.
- A brief discussion interpreting any trends you observe in the nucleus area and intensity data. Are there aspects of the pipeline could still be improved, and how would this affect these results?

6 Submission Instructions

Submit your assignment responses to Gradescope under Assignment 3.

As before, type up your solutions in the text editor of your choice. Please include the screenshots of your code for the corresponding questions in the write-up only for the functions we ask. Do not submit any code that has not been explicitly asked for.

Please tag all pages under the corresponding items; without this, we may not be able to give you all the points for a question.