Problem Set 3

Biological Physics

Due Thurs., Mar. 3, 2016

The Freely Jointed Chain (FJC) Model

A) Creating the Chains

For this section, you will write a basic simulation of the 2D Freely Jointed Chain model using Matlab.

1. Produce an $(N-1) \times m$ random matrix of angles, evenly distributed between 0 and 2π, where N is the number of monomers (use 200) and m is the number of simulations per polymer (use 500), and assign it to the variable `angles`. These will be the angles from the vertical for each bond. Each column will represent the bond angles from a different chain — there are $N-1$ bond angles in a chain of N monomers.

2. From this matrix of angles, produce a matrix of Δx and Δy for each bond in the matrix, for a bond length $b = 1$. You may want to use the `cos` and `sin` functions, which can be applied to matrices. Assign these to variables with appropriate names.

3. From the Δx and Δy matrices, produce two matrices with the x and y coordinates of the chains. You will want to use the `cumsum` function.

4. By using the `cumsum` function, we have now produced a matrix with the x and y coordinates of each monomer relative to the first monomer, but we have not put in the coordinates of the first monomer. The first monomer in each chain, therefore, should have coordinates (0,0). You should therefore add a row of zeroes to each of the two matrices, and reassign the output back to the same two variables. Remember that `[rowvector; matrix]` will produce a new matrix with the rowvector and the matrix concatenated.

5. **[Output]** Plot a graph of the first chain. Use `axis equal` to make the units of the x- and y-axes equal.

Now we should have two matrices, one of x-coordinates and another of y-coordinates, giving us the locations of each monomer in each of m chains. Let's take some statistics...

B) FJC Statistics

1. Let's find the average end-to-end distance for each of those chains. Remember that our matrices list coordinates as distances from the first monomer, so the end-to-end distance along the x-axis is just the last row of the matrix of x-coordinates, and similar for y-coordinates. Take these two rows out of their matrices, and put them in a new matrix with 2 rows, so that the first row is x-coordinates for each chain, the second is y-coordinates, and therefore the columns are the end-to-end vectors for each chain in the ensemble.

   (a) **[Output]** Find the average end-to-end vector $\left\langle \vec{R} \right\rangle$.

   (b) **[Output]** Find $\langle R^2 \rangle$. How does this compare to the theoretical result for the FJC?

2. Let's make a histogram and plot $P\left(|\vec{R}|\right)$. First, using your results from above, make a vector with $|\vec{R}|$ for each chain in it; then use the `hist` command to make a plot.

   (a) Use the `[n,xout] = hist(...)` form of the `hist` command to produce histogram data. The vector `xout` will hold the x coordinates for each of the bins (its $|\vec{R}|$ value), with `n` holding the number of chains that had that $|\vec{R}|$ value. The default in Matlab is to use 10 bins; you should make more than that, enough to show the general shape. Remember to use `doc hist` if you need help.

   (b) That gives you a count of the number of chains that have approximately each $|\vec{R}|$ value. To make a histogram for $P(|\vec{R}|)$, you should divide the `n` vector by the appropriate value; remember that $\int P(|\vec{R}|)\, d|\vec{R}| = 1$.

   Note: $P(R = |\vec{R}|)$ is a probability density, for which $P(R)\Delta R$ is the probability of a chain having an end-to-end distance within ΔR of R. In contrast, your `n` vector already has the ΔR built in. In step (3), you are going to compare your simulation results to the theoretical distribution we got in class, so you are going to want to make sure that the *area under* your probability distribution is equal to 1, not simply the sum.

   (c) **[Output]** Use the `stairs` function to plot the values above.

3. Plot the theoretical function $P(R) = \frac{2R}{Nb^2} e^{-\frac{R^2}{Nb^2}}$, on top of your histogram.

   (a) **[Output]** Use `plot` to overlay the theoretical result on top of the simulation results, with format `'k--'`. Use `xlabel`, `ylabel`, `title`, and `legend` to appropriately label your graph. Save your plot as a `.eps` file.

The Freely Rotating Chain (FRC) Model

For this section, you will analyze a simulation of the 3D Freely Rotating Chain model using Matlab.

The simulation has already been written for you; it is `frc.m` in the PS3 section of the Classes server. Download this file, and place it in the same directory where you are writing your own code for this problem set, and you can then use the `frc` function just as if it were a built-in function, and also call `help frc`.

The `frc` function produces a $3 \times N \times m$ matrix. The first dimension specifies the x, y, or z coordinate; the second, which monomer; the third, which configuration. Note that this output is the *separation vectors*, not the *location vectors*. I will refer to these as $\vec{r}_i$ to indicate the ith separation vector, i.e. $\vec{r}_i = \vec{R}_{i+1} - \vec{R}_i$.

C) Plotting a single chain

1. Use the `frc` function provided to produce a single chain of 200 separation vectors $\vec{r}_i$, with $\theta = \frac{\pi}{8}$. Calculate the locations of each monomer ($\vec{R}_i$).

2. **[Output]** Use the `plot3` command to make a 3D plot of this chain, with format `'.-'` (a line with points). Use the `axis equal` command to make the spacings on all three axes consistent with each other.

D) Analysis of $\langle \vec{r}_i \cdot \vec{r}_j \rangle$

1. Generate 1000 configurations of 200 monomers, for any value of θ you choose such that $0 < \theta < \pi$. From this $3 \times 200 \times 1000$ matrix, make a 200×1000 matrix of $|\vec{r}_i|$ values, and verify that they are one by taking the mean and standard deviation of this matrix.

2. **[Output]** Print out these values (the mean and standard deviation), and in your report, explain (very briefly) why they are what they are.
   Note: when computers do calculations using non-integer numbers, they do it imperfectly, and often have errors on the order of 10^{-16} or so.

3. **[Output]** Generate a 199×1000 matrix of $\vec{r}_i \cdot \vec{r}_{i+1}$ values, and find the mean and standard deviation as above. Print these two values, and also explain (very briefly) what the expected values should be.

4. **[Output]** Now plot $\langle \vec{r}_i \cdot \vec{r}_j \rangle$ vs. $|i - j|$, for θ in $\left\{\frac{\pi}{100}, \frac{\pi}{8}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}\right\}$, for both your simulation and the expected result from class.

   - First, I suggest you try it for just one value of θ, and then go back and make a `for` loop iterating over the values of θ above, using `hold all` to allow the colors to vary while several lines are added.
   - For each value of θ, you will need to use the `frc` function to generate a set of conformations.
   - Create an array of all possible values of $d = |i - j|$, and an array of zeroes (see `zeros`) of length N, where you can put your $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$ values.

- From here, create a `for` loop iterating over all possible values of $d = |i - j|$. Inside your `for` loop, make a matrix of $\vec{r}_i$ with $1 \leq i \leq N - d$ and a matrix of $\vec{r}_j$ where $1 + d \leq j \leq N$. Once you have these two matrices, you can element-wise multiply them and sum over the first dimension. This gives you a an $N \times m$ matrix of $\vec{r}_i \cdot \vec{r}_{i+d}$. Take the mean of the entire matrix (`mean(mean(M))`), and you have $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$, which you can then store in your $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$ array.
- Outside of the loop over d but inside the loop over θ, plot $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$ vs. d with points (`'.'`). Also, from what we learned in class, calculate the (exact) theoretical values for $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$ (remember $\langle \vec{r}_i \cdot \vec{r}_j \rangle = \cos(\theta)^{|i-j|}$) and plot it using a black dotted line (`'k:'`).
- Put a legend, title, and x and y labels on your graph. Use `xlim` and/or `ylim` to restrict your graph to an interesting region.

5. **[Output]** For each value of θ above, calculate the persistence length from your data and also the theoretical one, $\ell_p = \frac{-b}{\ln(\cos\theta)}$.

   - To calculate a persistence length from your data, use `find(tijs > exp(-1), 1, 'last')`, where `tijs` is your array for $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle$ as a function of d. The `tijs > exp(-1)` part generates a new array of booleans (0 or 1) indicating whether $\langle \vec{r}_i \cdot \vec{r}_{i+d} \rangle < \frac{1}{e}$ at that point, and the rest simply returns the last index (d) where that array is 1.

6. **[Output]** In your report, explain briefly (and abstractly, no math necessary) the change in the graphs from left to right as well as the differences between different θ values, and how these relate to the persistence lengths ℓ_p.

E) $\sqrt{\langle R^2 \rangle}$ vs. ℓ_p

In this section, we will find $\sqrt{\langle R^2 \rangle}$ for various values of θ (the bond angle), and then plot $\sqrt{\langle R^2 \rangle}$ vs. ℓ_p. We will then also plot the expected $\sqrt{\langle R^2 \rangle}$ that was calculated in class for the freely rotating chain, as well as that for the Worm-Like Chain model, and see how they compare. Use $N = 200$.

1. Generate an array `of` 20 values of ℓ_p logarithmically spaced from 10^{-2} to 10^4, and an array to contain $\sqrt{\langle R^2 \rangle}$. For each value of ℓ_p, calculate θ, and generate another 1000 configurations of 200 monomers, and calculate $\sqrt{\langle R^2 \rangle}$, where R^2 is the squared end-to-end distance, $\langle R^2 \rangle$ is the average over configurations, and N is the number of monomers. This can be done in much the same way as in Problem Set 1, except that you now have a 3-dimensional array, and you'll need to be more careful keeping track of which axis is which.
   As above, you may want a for loop here as well, filling in one $\sqrt{\langle R^2 \rangle}$ value for each iteration.

2. Plot $\sqrt{\langle R^2 \rangle}$ vs. ℓ_p using `semilogx`, which will give you a logarithmic x-axis for ℓ_p. Use `'r.'` to generate red dots at each relevant point. Use `ylim` to restrict the y-axis to a relevant region. Use the `hold all` command *after* the `semilogx` command; if you use it before, Matlab will not do a logarithmic axis as expected.

3. Now we want to graph this next to the theoretical results. Generate a new array of ℓ_p values with much smaller steps, calculate $\cos\theta$ from ℓ_p, and calculate both the Freely Rotating Chain expected results as well as those for the Worm-Like Chain from what you learned in class:

   - FRC: $\langle R^2 \rangle = Nb^2 \left(\frac{1+\cos\theta}{1-\cos\theta} - \frac{2\cos\theta}{N} \frac{1-\cos^N\theta}{(1-\cos\theta)^2} \right)$, where b is the bond length.
   - WLC: $\langle R^2 \rangle = 2\ell_\mathrm{p} R_\mathrm{max} - 2\ell_\mathrm{p}^2 \left(1 - \exp\left(-\frac{R_\mathrm{max}}{\ell_\mathrm{p}} \right) \right)$, where the maximum end-to-end distance is $R_\mathrm{max} = Nb$.

4. **[Output]** Plot these with different colors and different line-styles. Label your lines and axes.

5. **[Output]** In your report, explain where each of the two theoretical formulas for $\langle R^2 \rangle$ is accurate, and for areas where it is inaccurate, explain what approximation was made and why it does not work for that region of the graph.

Submission

As before, make a zip (or tar) file with your code in `.m` format, all your graphs in in `.eps` format, and the answers to the questions in `.txt`, `.rtf`, or `.pdf` format. Upload to the classesv2 server, *named with the format* ***`LASTNAME-FIRSTNAME-PS3.zip`*** (or `.tar` / `.tgz`).