Bioen 485/585 Lab 7a: Brownian Dynamics

In this lab, you will modeling a diffusion process and analyzing your results.

You will be reproducing part of the Nitta et al paper in which trajectories of microtubules are simulated on a microfabricated track. Because the orientation of the microtubule is important, the movement is modeled with a distance moved *r*, and an angle, θ, but you should still keep track of the position of the front of the microtubule in cartesion coordinates (x,y). At each step, both distance *r* and angle θ involve a deterministic and a stochastic fluctuation term. The distance moved, *r*, is a normally distributed random variable with mean Vav*dt and variance 2*Dv*dt (where Dv is the motional diffusion coefficient.) The new angle is θ+dθ, where dθ is normally distributed, with mean 0 and variance Vav*dt/Lp (where Vav is the average velocity, and Lp is the persistence length of the trajectories.) The new position will then be (x+dx, y+dy) where dx = *r**cos(θ) and dy = *r**sin(θ), using the new θ. According to the paper, a time step of dt = 0.1 sec is appropriate. For convenience, you should start with an initial position of (0,0) and an initial angle of 0 (heading straight out along the x-axis). The paper uses the following parameter values, which were determined experimentally:

- Dv = 2e-3 um^2/sec is the motional diffusion coefficient
- Vav = 0.85 um/sec is the average velocity
- Lp = 111 um is the persistence length of the trajectories

1. (20 points) **Method and verification**. Simulate the motion of a single microtubule without any microfabricated tracks, and verify your work. *Hint: if you find a mistake using these verifications, determine whether it is your coding mistake or whether the assumptions in the paper were inappropriate for the data they seek to reproduce. If this is the case, you can go on to parts 2 and 3 using their method or a fixed method you devise.*
    a. Include a plot that shows several sample trajectories (plotting x vs y, not x vs time and y vs time). *Hint: You will need to write an algorithm to numerically simulate a Wiener Process by integrating (numerically solving) an SDE (stochastic differential equation) that contains a deterministic and a stochastic term. Hint: look at the code on the top of page 6 of the "lab 0" tutorial for how to write your own simple algorithm to solve a deterministic ODE instead of using a solver. Your code will include both deterministic and stochastic terms. You can pick how long (how many seconds) to run the simulations. You only need to plot enough trajectories to get an idea of the types of data that result; too many will cause them to obscure each other. For the remaining verifications, you will want to run enough trajectories to get statistics. You decide how many for all problems.*
    b. Calculate Vav and Dv from your trajectories and compare to the parameter value you used. *Hint: calculate the contour length s to get total distance moved (the length of the path, not the end-to-end distance) in the time allowed. What value do you expect for the mean and standard deviation of s, based on the definitions and values of the parameters? From the lecture notes, you learned how the square of the distance diffused ($\langle x^2 \rangle$) can be predicted from the diffusion coefficient. Here, the 'distance diffused' is $\sqrt{(S - \langle S \rangle)^2}$.*

   c. Calculate the persistence length, Lp, in your trajectories, and compare this to the parameter value you used. *Hint: the definition of Lp is* $\langle cos\theta(s)\rangle = e^{-s/L_p}$*, where* θ *is the change in angle between the tangent to the beginning and end of a segment, and s is the contour length as defined above. So, calculate the average of* θ *over all tracks at each time point, and plot this against the expected value of s at each time point. Is this data consistent with the exponential decay* e^{-s/L_p} *with the value of Lp that you used in the simulation?*
   d. Ensure that your results are independent of the time step used, providing whatever figures or calculations you need for this verification. *Hint: if you run with a smaller time step, do you get similar values of Vav, Dv, and Lp? If you have a major bug, it will be obvious. Don't worry about subtle statistical differences.*
2. (10 points) **Validation.** Simulate the motion of a single microtubule within a 5.5 μm-wide microfabricated track.
   a. Plot the walls and one or more sample trajectories to check for bugs and to illustrate typical behaviors. *Hint: check the position at each time step. If the y-position is outside the channel wall, bring it back to the channel wall and set the angle to be 0, so it is moving parallel to the wall. Then calculate the x-position to match the required distance moved for that step.*
   b. Calculate the average distance between collisions and the average collision angle in these conditions, and validate this against the experimental data provided in the paper. Do any of your microtubules turn around? *Hint: a collision occurs when the microtubule tries to go outside the channel walls, as described above. Collisions were only counted for multiple collisions if microtubules traveled at least 0.5 μm away from the boundary or traveled more than 20 μm along the boundary. The paper notes that a 5.5 μm wide channel had collisions every 24 +/- 14 um with collision angles 28 +/- 21°. Because these are stochastic, you won't get the exact same thing, but it should be comparable. Make sure you run long enough distances to not be affected by the initial conditions. Indeed, you may want to run one very long track to solve this problem as there is no advantage of running many tracks in parallel and it makes tracking the data harder.*
3. (10 points) **Value added: Design**. Use your simulations to test a design of some sort of nanochannel structure or device. *Hint: there are several structures/devices in the paper, such as rectifiers and concentrators. The biggest challenge of these will be to code the geometrical constraints defining a collision. If this sounds like fun, go for it. Otherwise, you may want to ask a simpler design question, such as how wide you can make a channel without the microtubules switching directions. (You will need to describe a measurement of frequency of direction switching per contour length, or something.)*

Additional hints for counting collisions:

1. If you just run one track for a long time, you can track the distance from the last collision, as well as the maximum distance moved since the last collision from the nearest wall, to see if you are ready for another collision. You can also keep an array of all the collision angles and distances since last collision, so you can take mean and

standard deviation on these values at the end of the simulation. If you want to restart your track when the microtubule reverses direction, for example, you can always do this and keep building on the same arrays. You should be able to do all this using arrays and scalars.

2. As an alternative, you may want to use a structure to keep track of data organized by track. A structure allows you to keep structured data that includes arrays of different lengths or different types of data, such as strings versus scalars versus arrays. You can easily make an array of structures as illustrated below, where n is the index into the array. If you haven't programmed in MATLAB with structures, start by reading some examples after typing doc structure into the command line. For example, one way I solved this problem was to define a "collisions" structure as follows, to keep track of the list of angles, list of distances for each track, as well as the number of collisions on that track and the distance since the last collision.

```
collisions(n).num = collisions(n).num +1;
% on the bottom border, angles are negative
if Y(n,t) < 0; Q(n,t) = -Q(n,t); end
collisions(n).angles(end+1) = Q(n,t);
collisions(n).distances(end+1) = collisions(n).s;
collisions(n).s = -dr(n);
collisions(n).maxy = 0;
```

(dr is the distance moved and Q(n,t) is the angle moved in the t^{th} step of the n^{th} track.) The advantage of the structure is that each element of an array of structures (here, the collision structure for the nth track) can have arrays of different lengths. If you tried to build a large matrix to store arrays of varying length, it won't work. Also, the names in the structure, while longer to write, are descriptive to help you remember and understand your variables. Finally, it is easy to transfer an entire set of structured data into and out of a function with a single variable name.

After I fill the structures, I can do things like concatenate the arrays of angles into one long array if I want:

```
[collisions.angles]
```

3. Now for some hints for figuring out when collisions occurred
   a. If you run a channel from –w/2 to w/2, where w is the width of the channel, and Y is the y-position of the track after the projected step, then you can test for a possible collision with
      `if abs(Y) > w/2`
   b. if you are tracking the distance and maximum y-displacement from a channel wall as described in the structure above, you can test for whether the requirements of a collision relative to the last collision are met:

```
if (collisions(n).s > 20 || collisions(n).maxy > 0.5)
```

   c. remember that if Q is the angle of the track, then the angle between the track and the wall is +Q if you collide with the upper wall, but –Q at the lower wall.