

PHY307, Science and Computers I
Monte Carlo Lab #2, November 12 and 14, 2002
(due by end of lab Thursday)

Stumbling ethanol abusers: alone and with friends

Summary of what you will do: Many physical, biological, chemical, and even economic processes are modeled with random walks. Small particles can be rapidly affected by collisions from molecules move in a random fashion. This is sometimes referred to as a “drunkard’s walk”. These particles might be atoms, molecules, or even living cells. The position of these particles steps in random directions over time. As the processes that affect the price of financial instruments are unpredictable, the steps in price are also often modeled as random.

In the first part of this lab, you will model such random walks.

In the second part of the lab, you will look at random aggregates. Particles will execute random walks until they stick to an aggregating mass. These aggregates can have a very low density: you will study how that density changes as particles aggregate and deduce a fractal dimension for the structure.

In lecture segments, we will discuss random walks and look at movies of random walkers and pictures of aggregates.

LAB REPORT

1. Get set up. Start a lab report entitled “*MyLastNameWalkers.doc*”, where *MyLastName* is your actual last name.
2. Write a module, **walk1D.py**, which will simulate the motion of a random walker in one dimension (drunk on a sidewalk) and plot the x -coordinate as a function of time. The walker itself will be a sphere and a cylinder attached to a frame. This frame will move along the x -axis by random amounts.
 - a. Import the modules you will need: **visual**, **visual.graph**, and **random**.
 - b. Set **scene.autoscale** to 0.
 - c. Set a time variable, **t**, with initial value 0.
 - d. Make a frame: this frame, holding the sphere and cylinder, will “walk”.
 - e. Create a cylinder, with **axis=(0,2,0)**, attached to the frame. Create a sphere, attached to the frame with **pos=(0,3,0)**.
 - f. RUN YOUR PROGRAM to see if it works so far.
 - g. Add in a plot, using **gcurve()**, as in, **myplot = gcurve()**
 - h. Now set up a for loop to run 10,000 time steps. In this loop:
 1. Keep the rate slow enough, say **rate(40)**.
 2. Add a random step to the x -coordinate of the frame. Do this by adding **random()-0.5** to the frame’s **pos.x** attribute.
 3. Plot the new position as a function of time. For example, **myplot.plot(pos=(t,name_of_frame.pos.x))**
 4. Add 1 to your time variable.
3. Include your code and a snapshot of your plot in your report. Comment on the appearance of the line in your plot.

4. Start a new code, **manywalkers3D.py**, that will simulate many particles, moving in three dimensions like fragrance molecules in still air or like fat globules in milk:
 - a. Import **visual** and **random**.
 - b. Turn off autoscaling.
 - c. Start an empty list, named **particles**.
 - d. Use a **for** loop over, **for j in range(300):**, to make 300 spheres, each appended in turn to the list **particles**. Give each sphere a radius of 0.5.
 - e. RUN YOUR PROGRAM to make sure it works so far.
 - f. Now add on the dynamics for the particles. In an infinite **while 1:** loop,
 1. Use a **for** loop to loop over all of the spheres in the list **particles**. Add a vector in a random direction to the spheres position. You can do this by adding `random()-0.5` to each of the x, y, and z positions, for example.
 - g. Run your program – record your program and what you see.

5. Now you will augment **manywalkers3D.py** to make a plot of the average of the square of the distance of the particles from their starting point, as a function of time.
 - a. Save a copy of your code to **manywalkers3Dplot.py**.
 - b. Import **visual.graph** at the beginning of your code.
 - c. Make a curve, **c = gcurve()**, near the beginning.
 - d. Set a time variable to zero (**t=0**), near the beginning.
 - e. In the while 1: loop, you want to compute the average of the square distance. This is what you want to add (sets **total** to **0**, adds the square of the magnitude of the particle position to the total for all particles, then divides by the number of particles and plots versus time **t**):

```

total = 0
for particle in particles:
    total = total + mag2(particle.pos)
total = total/len(particles)
c.plot(pos=(t, total))
t = t+1

```

- f. Run this program. Include this code in your report and comment on the shape of the plot you get.

AGGREGATION OF STICKY DRUNKS

Please read the paragraph and instructions carefully and slowly, so that you understand what you are doing.

For this last part of the lab, you will study aggregating random walkers. The simulation will have a growing aggregate, represented in python as a list of spheres. Each new sphere will be created near the aggregate. This new sphere will walk in random directions until it collides with the aggregate (or gets too far away, in which case it will be reset near the aggregate again.) When a collision occurs, the sphere will stick to the aggregate.

This process mimics the aggregation of particles in suspensions, some biological growth processes, and is similar to how lightning strikes develop.

Note that the function `len()` can be used to find the “length” of objects – in the case of a list named `mylist`, for example, `len(mylist)` gives an integer which tells how many items are in the list.

6. Download the code `DLA_3D_class.py`.
7. The code includes a function `nocollision`. Explain how this function works, line by line.
8. The second function is `randomdirection()`. This function returns a vector that has length 1 and points in a random direction. (This is a more correct choice of random direction. The method we have used before is biased towards the corner of the cube. Here, we really need a totally random direction.) Explain how this function works, line by line:
 - a. Note that the `mag` function takes a vector as an argument and returns the length of the vector. The `random()` function returns a number from 0 to 1.
 - b. What does the `while` loop check? Note that the first line of the function just sets `d` as a vector for which this condition is true, so that a random `d` will be generated.
 - c. Once a “proper” vector `d` is generated, one that satisfies the condition in the `while` statement, what happens in the last line of the function?

This function, `randomdirection()`, will be used both to generate the initial location of new particles and to move them around randomly.

9. The rest of the code can be described in outline form. Please review how this works:
 - a. Get the size of the aggregate to be built.
 - b. Initialize the aggregate with one blue sphere.
 - c. While the aggregate needs growing:
 1. Make a new green sphere near the aggregate (at distance **aggregateradius** from the origin.)
 2. While the new sphere is not touching the aggregate, it walks in a random direction.
 3. Once it collides, append it to the list **aggregate**, change its color to red.
 4. Check if the sphere has increased the radius of the aggregate. Do this by comparing the distance from the origin of the new sphere (+1 as the sphere sticks out a distance 1 from its own center) with the current radius of the aggregate.
 5. Print out current mass and size of aggregate for this step.
 - d. The program wraps up by reporting the size of the aggregate as a final result.
10. Modify the program to report, in addition to the radius, the density of the aggregate. This will be done by adding a single line at the end of the program. Estimate the density by dividing the mass by an estimate of the volume of the aggregate.
11. Run the program for aggregate masses of 2, 10, and 50. Save snapshots of these aggregates to your report.
12. Run the program 6 times each for sizes of 5, 10, 20, and 40, recording the aggregate radius and density each time, in a table. Note trends in the average radius and density with aggregate mass. Run the program once for size 80, again recording the radius and density.
13. We will average the class data to get a data set of mass versus radius, which, when plotted on a log-log plot, will give an estimate of the fractal dimension of diffusion limited aggregates.
14. If you have time, for extra credit, modify your code to run for 2D: just set **randomdirection()** to generate points that have random x and y coordinates, but with zero z-coordinate. Estimate the density by dividing the mass by the area of a circle with radius of the aggregate. How does the density depend on size? Can you estimate the fractal dimension of aggregates in 2D?