PHY407 Formal Lab Report 4

Fall 2024

Background

Diffusion-Limited Aggregation (DLA): A "random walk", also called "Brownian Motion", describes the motion of a particle (such as a smoke or dust particle) in a gas, as it is buffeted by random collisions with the gas molecules. In the simplest version of DLA, a particle is confined to a box, in which it performs a random walk until it reaches one of the sides. At that point, it sticks to the side, becoming anchored there and immovable. (See Fig 1 for an example of a particle's path.) Then a second particle is added to the interior of the box and does a random walk, until it sticks either to a side or to the other particle. Then a third particle is added, and so on.

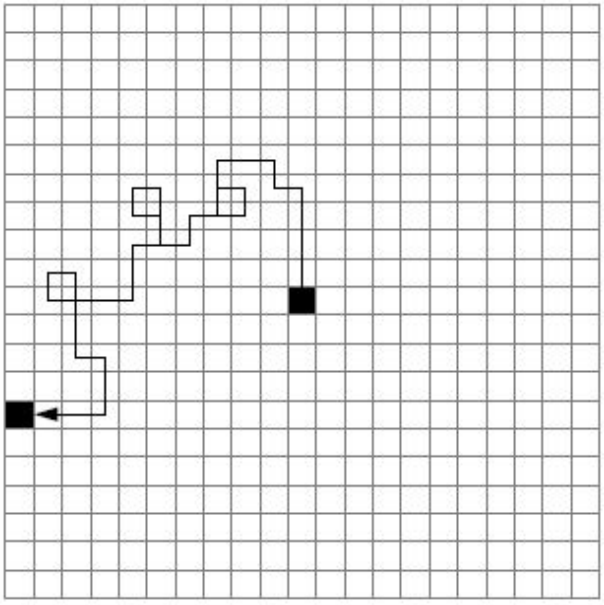

Figure 1: Example of a particle's path in DLA.

Simulated Annealing: For a physical system in equilibrium at temperature T, the probability that at any moment the system is in a state i is given by the Boltzmann probability

$$P(E_i) = \frac{\exp(-\beta E_i)}{Z}, \tag{1}$$

with $Z = \sum_i \exp(-\beta E_i)$ and $\beta = \frac{1}{k_B T}$. Assume the system has a single unique ground state and choose the energy scale so that $E_i = 0$ in the ground state and $E_i > 0$ for all other states. As the system cools down,

$T \to 0$, $\beta \to \infty$,$\exp(-\beta E_i) \to 0$ except for the ground state where $\exp(-\beta E_i) = 1$. Thus in this limit $Z = 1$ and

$$P_a = \begin{cases} 0, & E_i = 0 \\ 1, & E_i > 0 \end{cases} \tag{2}$$

This is just a way of saying that at absolute zero, the system will definitely be in the ground state.

This suggests a computational strategy, "simulated annealing", for finding the global minimum of a function: set up a system corresponding to the function, simulate the system at temperature T using the Markov Chain Monte Carlo method, then lower the temperature to 0 and the system should find the ground state. This approach can be used to find the global minimum of *any* function f by treating the independent variables as defining a 'state' of the system, and f as being the energy of that system. There is one issue that needs to be dealt with: If the system finds itself in a local minimum of the energy, then all proposed Monte Carlo moves will be to states with higher energy and if we then set $T = 0$ the acceptance probability becomes 0 for every move so the system will never escape the local minimum. To get around this, we need to cool the system slowly by gradually lowering the temperature rather than setting it directly to 0.

For efficiency of simulated annealing, pick the initial temperature such that $\beta(E_j - E_i) \ll 1$ meaning that most moves will be accepted and the state of the system will be rapidly randomized no matter what the starting state. Then choose a cooling rate, typically exponential, such as

$$T = T_0 \exp\left(-\frac{t}{\tau}\right) \tag{3}$$

where T_0 is the initial temperature and τ is a time constant. Some trial and error is needed in picking τ. The larger the value, the better the results because of slower cooling, but also the longer it takes the system to reach the ground state.

Questions

1. Simulating the DLA Process

(a) Make a simple computer simulation of Brownian Motion of a particle in two dimensions, as follows:

- Discretize the space in which the particle will move, by creating a square $L \times L$ grid such that the particle's position can be represented by two integers $i,\ j = 0 \ldots L-1$. (Assume an odd value for L, so there is one grid site exactly in the center.)
- Begin the simulation with the particle on the center grid site ($x = y = 0$) at $t = 0$.
- At each time-step of the simulation, choose a random direction (up, down, left, or right) and move the particle one grid space in that direction. The particle is not allowed to move outside the limits of the grid - if it tries to do so, choose a different random direction to move instead.
- Perform this process for N time-steps.

Run your simulation with $N = 5000,\ L = 101$. Assuming each time-step is $\delta t = 1$ ms long, and the grid spacing is $\delta x = \delta y = 1$ mm: make a plot of the particle's x position vs t, a plot of y position vs t, and a plot of y position vs x position.

Submit your code and plots.

(b) Incorporating some of your code from the previous part, create a program to simulate the DLA process for n particles in a 2-dimensional box. Ensure the spatial grid spans the box, and introduce each new particle at the center grid site. Have the program stop once there is an anchored particle at this center grid site, and display a figure showing the positions of all the particles at this time-step.

Hint: pay attention to how you store the positions of the anchored particles, and how you check at each time-step each particle's neighboring squares to see if they are outside the box or are occupied by an anchored particle.

Submit your code and figure.

2. Efficiency of Simulated Annealing

(a) Consider the function

$$f(x, y) = x^2 - \cos(4\pi x) + (y - 1)^2. \tag{4}$$

The global minimum of this function is at $(x, y) = (0, 1)$. Write a program to confirm this fact using simulated annealing.

- Start at $(x, y) = (x_0, y_0)$ where x_0 and y_0 are parameters that can be set in your code.
- Use moves of the form $(x, y) \to (x + \delta x, y + \delta y)$, where δx and δy are random numbers drawn from a Gaussian distribution with mean 0 and standard deviation 1.
- Use an exponential cooling schedule, with the initial temperature T_0, final temperature T_F, and time constant τ as parameters that can be set in your code.

Using $x_0 = y_0 = 2$, run your code repeatedly, adjusting the 3 parameters of your cooling schedule (T_0, T_F, τ) until you find values that reliably give the correct answer in reasonable time. Now have your program plot the values of x and y as a function of time-step, and print out the final value of (x, y) at the end. *Hint: You will find the plot easier to interpret if you make it using dots rather than lines.*

Submit your plots, the values of (T_0, T_F, τ) you used, and printout of the final value of (x, y).

(b) Now adapt your program in Part (a) to find the minimum of the more complicated function

$$f(x, y) = \cos x + \cos(\sqrt{2}x) + \cos(\sqrt{3}x) + (y - 1)^2 \tag{5}$$

in the range $0 < x < 50$, $-20 < y < 20$. This means you should reject (x, y) values outside these ranges. *Hint: the correct answer is around $x \approx 16$ and $y = 1$, but there are competing minima for $y = 1$ and $x \approx 2$ and $x \approx 42$, so if the program settles on these other solutions it is not necessarily wrong.*

As in Part (a), your program should make a plot of the values of x and y as a function of time-step, and print out the final value of (x, y) at the end.

Submit your plots, the values of (T_0, T_F, τ) you used, and printout of the final value of (x, y).

3. Importance Sampling

(a) Consider:

$$\int_0^1 \frac{x^{-1/2}}{1 + e^x} dx. \tag{6}$$

Note the integrand has a divergence. Evaluate this integral using the mean value method with 10,000 sample points. Repeat this calculation 1000 times, and find the mean of your 1000 results. *Hint: it will take awhile, on the order of a minute, to run 1000 times - so first debug your code using a smaller number of repetitions.*

Submit your code, and printout of the mean.

(b) Now evaluate this integral using the importance sampling method. Repeat this calculation 1000 times, and find the mean of your 1000 results. *Hint: use the weighting function $w(x) = x^{-1/2}$, which removes the singularity. You will need to determine the probability distribution $p(x)$ from which you should sample, and the transformation $x(z)$ that should be applied to uniformly-distributed random numbers z in order to sample from this distribution.*

Submit your code, printout of the mean, and brief written explanation showing how you found $p(x)$ and $x(z)$.

(c) For each of the two methods separately: make a histogram of the 1000 results you obtained, with 100 bins over an appropriate range. (You probably want to use `matplotlib.pyplot.hist`.) Comment on what your histograms tell you about each method.

Submit your histograms and brief written answer.

(d) Importance sampling also helps evaluate rapidly varying integrands even if they aren't singular. For example, consider the integral

$$\int_0^{10} \exp(-2|x-5|)dx. \tag{7}$$

The integrand is sharply peaked near $x = 5$. If we calculate the integral using points that are uniformly-distributed on the interval 0 <= x <= 10, we will be using a lot of points that don't significantly contribute to the result. To sample more points near $x = 5$, thereby increasing the precision of the sampling method, use importance sampling with weighting function

$$w(x) = \frac{1}{\sqrt{2\pi}} e^{-(x-5)^2/2}. \tag{8}$$

Since $w(x)$ integrates to almost exactly 1 over the relevant interval, you can take $p(x) = w(x)$, and use `numpy.random.normal` to sample from $p(x)$. Using 10,000 sample points, repeat the calculation of the integral 1000 times, and histogram the results (with 100 bins over an appropriate range).

Submit your code and histogram.