Statistische Physik im Gleichgewicht

WS 2023/2024 – Blatt 2

Dr. Milos Knezevic, Dr. Jeanine Shea Abgabe 6.11.2023

Problem 3: Biased 1D Random Walk (*4 Points*)

[C] We repeat the random walk from Problem 2, but this time in 1D. Therefore there are only two possible neighbor sites to randomly choose in each step.

(a) Repeat the random walk, this time in one dimension. Again, plot the mean end-to-end distance as a function of N (for $N \in 10, 20, 30, ..., 980, 990, 1000$) with an appropriate number of runs.

(b) Modify the random walk function, so that the probabilities for going left and right are not 50-50 but can be changed. This is called a biased random walk. You may change input, output and the function itself in any way that seems helpful to you.

(c) Perform biased random walks with $p_r = 0.7, 0.8, 0.9$, where p_r is the probability to move to the right. Calculate the mean end-to-end distance with an appropriate number of runs and plot it as a function of N (for $N \in 10, 20, 30, ..., 980, 990, 1000$). What changes and why?

Problem 4: Nonreversal Random Walk and Self-avoiding Walk (*6 Points*)

[C] Now we go back to the random walk in two dimensions. We want to modify it slightly, such that we simulate a nonreversal random walk.

(a) Modify the 2D random walk on a grid, such that it is forbidden for the algorithm to immediately go back to the same location it just came from. It is still allowed to cross its own path and to retrace a path it has been on before.

(b) Measure the mean square end-to-end distance as a function of N (for $N \in 10, 20, 30, ..., 980, 990, 1000$). Plot the results, determine the slope and compare it to the standard random walk.

As a second step, we consider a self-avoiding walk. It is a form of random walk, where points that were visited before are never visited again, i.e. the trajectory can not intersect with itself. Thus, we can think of this as a model polymer in two dimensions with a random configuration.

(c) Implement a self-avoiding walk. This function should take the maximum number of steps as input. But, for the self-avoiding walk, it is not guaranteed that the maximum number of steps is realized, because the trajectory can get stuck. As soon as that happens, the function should end the computation. On the one hand, it should return a two-dimensional numpy array with the x and y positions of every step. It should also return the actual number of steps performed by the walk. Perform self-avoiding walks until you have 5 that reached $N = 50$ and plot their trajectories (plot separately to show that they indeed do not cross their own path). Provide the number of trials that you needed.

Feedback:

Roughly how much time did you spend on this problem set?