Homework 5

Akash

Problem 1

In the forst problem to implement the random Walk, I used tensorflow module. To get the random element form the given set [-1,0,1], I used **tf.random.uniform** function. for the remaining part I was using **tf.math** module which is nearly same features as numpy.

My goal was to get the slope of $\sigma_n^2 vsn(numberofsteps)$ for 1D,2D and 3D. by the theoretical prediction it supposed to be **1 for 1D**, **2 for 2D** and **3 for 3D**. also we tried to diffusion constant for all the three dimention which is theoretically $\mathbf{D = 1/2}$.

In this random walk, I tried many such walkers and took the ensamble average of it, which basically same as saying single walker will move in steady state according to ergodic principle. That's exactly what we are trying to do here to het the empirical result.

In 1D we got slop for the $\sigma_n^2 vsn$ value as o.62 in tensorflow as I used only 80 walker to average over. while the Diffusion constant $ D = slope/2 D = 0.31$. while comparing with the Numpy i got $ D = 0.5$, which is close to the theoretical prediction as I have taken 1000 walkers average, which is statistically good sample average in this case.

In 2D we got slop for the $\sigma_n^2 vsn$ value as 1.86 in tensorflow as I used only 80 walker to average over. while the Diffusion constant $ D = slope/4 D = 0.465$. while comparing with the Numpy i got $ D = 0.465$, which tells us that the tensorflow converged to the answer way quickly compare to numpy. This is just a obseravation not conclusion.

In 2D we got slop for the $\sigma_n^2 vsn$ value as 2.83 in tensorflow as I used only 80 walker to average over. while the Diffusion constant $ D = slope/6 D = 0.47$. while comparing with the Numpy i got $ D = 0.46$, here as well we can see as we go higher dimention converging in tensorflow is better. I don't know what exactly is the reason behind that.

Problem 2

I couldn't install the tensorflow-probability. so i decided to do this problem in google colab. I also implemented the code in numpy as well. I ran this code on 50000 steps to get the probability distribution I was sampling form. In this I generated the probability distribution using **tfp.distributions.mixture**.

Now for the random sampling I was using **tfp.mcmc.RandomWalkMetropolis**. and then using function **tfp.mcmc.sample_chain** I got the markov chain which ended up stationary in the end. I removed first 1000 steps from the chain as they were not the samples from the distribution.