Markov Chain Monte Carlo

Question 1 (Sampling from Permutations)

1. Suppose we want to generate a uniformly distributed element in $\mathscr{S}$, the set of all permutations of $(x_1, x_2, \cdots, x_n)$ of the numbers $(1, 2, \cdots, n)$ for which $\sum_{j=1}^{n} jx_j > a$ for a given constant a.

2. (**R**) Start with $n = 4$ and $a = 14$. List all the feasible permutations. Write a small python program to generate a uniformly random permutation just for this example. Run this program sufficiently large number of times and plot a histogram of frequencies of feasible permutations.

3. (**R**) Now let us do the same exercise using MCMC. In the Markov chain modeling the state space is given by all possible feasible permutations. We will first write a function that generates a (uniformly) random feasible neighbor of a given permutation. Two permutations in $\mathscr{S}$ are said to be neighbours if one result from an interchange of two of the positions of the other, i.e., $(1, 2, 3, 4)$ and $(1, 2, 4, 3)$ are neighbours but $(1, 2, 3, 4)$ and $(1, 3, 4, 2)$ are not. Now find feasible set of neighbours among all neighbours. Implement this function to generate (uniformly) random feasible neighbor of a given permutation. Each permutation is a state of the Markov chain. Note that this probability is denoted as $q(i, j)$, representing the probability of jumping to state j from current state i. $q(i, j) = 1/|\mathcal{N}(i)|$, for $j \in \mathcal{N}(i)$, the set of neighbours of i.

4. Find
$$\alpha(i, j) = \min\left\{\frac{\pi(j)q(j, i)}{\pi(i)q(i, j)}, 1\right\}$$
Note that $\pi(s)$ is same for all feasible permutations. After choosing a neighbour with probability $q(i, j)$, the next state will be j with probability $\alpha(i, j)$.

5. (**R**) Write a program to generate the required permutations using MCMC approach. Does it give reasonable output? Experiment with your program and try several different values of a and n. By picking some trivial values of a check if your MCMC implementation is giving the expected results.

Question 2 (Bayesian inference)

1. For this exercise we will see how MCMC can be used in Bayesian inference. We would like to find the most likely distribution of θ, the parameters of the model explaining the data, D. Here we are mostly interested in the specific formulation of Bayes formula:
$$\mathbb{P}(\theta/D) = \frac{\mathbb{P}(D/\theta)\mathbb{P}(\theta)}{\mathbb{P}(D)}$$
$\mathbb{P}(\theta/D)$ is the posterior distribution, $\mathbb{P}(D/\theta)$ is the likelihood, $\mathbb{P}(\theta)$ is the prior and $\mathbb{P}(D)$ is called the evidence. Computing some of these probabilities can be tedious, especially the evidence $\mathbb{P}(D)$. Here we will use MCMC, which will allow us to sample from the posterior, and a draw distributions over our parameters without having to worry about computing the evidence.

2. Generate 1000 samples from a normal distribution with mean $\mu = 10$, and standard deviation $\sigma = 3$. Plot the histogram.

3. We would like to find a distribution for $\sigma_{observed}$ using observed samples. For now we will assume that $\mu = 10$ is known. Let $\theta = (\mu, \sigma)$ and μ is known. For the prior, only assume that σ is positive. So, $\mathbb{P}(\theta) = 1$ if $\sigma > 0$ where $\theta = (\mu, \sigma)$. Create a python function $prior(\theta)$ which reflects that.

4. For the Markov chain construction consider $\theta = (\mu, \sigma)$ to be the the states which, is same as taking σ as the state. The transition model is given by $Q(\sigma_{new}/\sigma_{current}) = \mathcal{N}(\sigma_{current}, 1)$. Create appropriate python function.

5. Now we will consider the likelihood function. The likelihood is defined as following,

$$\mathbb{P}(D/\mu_{obs}, \sigma_a) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_a^2}} e^{-\frac{(d_i - \mu_{obs})^2}{2\sigma_a^2}}$$

where d_i is data point in D, $\mu_{obs} = \mu$ in known (as of now) and $a = new$ or $current$. We will take logarithm on both sides and define log-likelihood as

$$\mathbb{L}(D/\mu_{obs}, \sigma_a) = \sum_{i=1}^{n} \log \left[\frac{1}{\sqrt{2\pi\sigma_a^2}} e^{-\frac{(d_i - \mu_{obs})^2}{2\sigma_a^2}} \right]$$

Explain why taking logarithm is helpful. Write a function in python that returns $\mathbb{L}(D/\mu_{obs}, \sigma_a)$.

6. The acceptance probability is given by

$$\alpha(\theta_{current}, \theta_{new}) = \min \left\{ 1, \frac{\mathbb{P}(D/\theta_{new})\mathbb{P}(\theta_{new})}{\mathbb{P}(D/\theta_{current})\mathbb{P}(\theta_{current})} \right\}$$

where $\theta_{new} = (\mu, \sigma_{new})$ and $\theta_{current}$ is defined similarly. Note that we have access to the log-likelihood $\mathbb{L}(D/\theta)$, so make appropriate changes for that in the acceptance probability. Write a function *acceptance*($\theta_{current}$, θ_{new}) that return True or False according to the new sample is accepted or not.

7. Proceed similarly as you do in generic Metropolis algorithm. Start with initial state $\theta_0 = (10, 0.1)$. Run the MCMC for 25000 iterations and keep track of accepted and rejected σ values. Plot both accepted and rejected σ values against iteration for the first 200 iterations. Then plot the same for all iterations. Explain your observations.

8. Discard the first 25% of the σ values which are accepted. Plot the histogram of the remaining σ values. Explain your observations. Why dropping first few σ values makes sense?

9. Now assume that μ and σ are both unknown. Same technique will be used to estimate μ and σ now. We have to change the prior first. Assume that μ and σ are independent and μ is uniformly distributed between 5 and 15. $\mathbb{P}(\theta) = \mathbb{P}(\mu)\mathbb{P}(\sigma)$ with $\mathbb{P}(\sigma)$ as before. The state space is θ and the transition probabilities are given by $Q(\theta_{new}/\theta_{current}) = \mathcal{N}(\theta_{current}, \mathbf{I})$, which is now a bivariate normal distribution. Make similar changes in other quantities like μ_a in place of μ_{obs} in the likelihood function, $\theta_{new} = (\mu_{new}, \sigma_{new})$, $\theta_{current}$.

10. Start with initial value $\theta_0 = (5, 0.1)$ and run the algorithm for some suitable iterates. After discarding first 25% of accepted θ, plot separate histograms for μ and σ.