The Metropolis Algorithm

1 Sampling Probability Distributions

We often encounter the following situation: given a probability distribution $\pi(\mathbf{x})$ over a (vector) random variable $\mathbf{x} = (x_1, x_2, .., x_D)$ one wishes to compute the mean value of a function $f(\mathbf{x})$. The *State Space* S of $\mathbf{x}$ is the set of possible values that $\mathbf{x}$ can take (which could be finite or infinite). For this analysis, it is useful to assume that the set S is discrete, $S = \{\mathbf{x}_i\}$. Then,

$$\bar{f} = \sum_i f(\mathbf{x}_i)\pi(\mathbf{x}_i) \tag{1}$$

Often, this average cannot be computed analytically, and needs to be computed numerically. Then, one can in principle generate this distribution on the computer and calculate the average. One way to go about it is to generate the various values $\mathbf{x}_i$ with the corresponding probabilities $\pi(\mathbf{x}_i)$. Then, if $\mathbf{x}_i$ has been generated M_i times, the average of f is given by

$$\bar{f} = \frac{\sum_i M_i \times f(\mathbf{x}_i)}{M}$$

where $M = \sum_i M_i$. This can also be directly written as

$$\bar{f} = \frac{1}{M}\sum_r f_r \tag{2}$$

where here r refers to a 'trial' in which the variable $\mathbf{x}$ is generated with probability $\pi(\mathbf{x})$, f_r is the value of f for that trial and M is the total number of trials. In this expression, a given value f_r will usually occur multiple times, in proportion to the probability $\pi(\mathbf{x}_r)$ of $\mathbf{x}_r$ occurring.

To sample such probability distributions and compute averages, we can in principle use Bayes Theorem and a random number generator which generates random numbers uniformly between 0 and 1. This technique is called 'direct sampling', since it directly samples the probability distribution. For probability distributions over one or two variables, this method works well. However, for probability distributions over several variables (for which the vector $\mathbf{x}$ has a high dimension), this is usually impractical, since the computing time increases exponentially with the dimension of $\mathbf{x}$.

2 Monte Carlo Sampling

Monte Carlo sampling involves taking a random walk in the space of possible values of $\mathbf{x}$, a walk which is generated such that after a 'large enough' number of steps of the walk, different values of $\mathbf{x}$ have been 'stepped-on' by the virtual walker a number of times proportional to the probability with which they occur in the distribution $\pi(\mathbf{x})$. Such a probabilistic random walk is an example of a *Markov Chain*. To visualise this, it is convenient to imagine the space of values of $\mathbf{x}$ being divided into cells. The walker hops from one cell to another with a certain probability, such that any one cell is connected to others by a set of probabilities with which it steps on them. This set of probabilities forms the algorithm which eventually covers the cells with the overall probability $\pi(\mathbf{x})$.

Say, there are N cells and the walker starts in cell 1. Then, with probability $P(1 \longrightarrow i)$, he steps on the i^{th} cell. Let us write the various cell probabilities as a column vector. Then, at the beginning, the probability vector will have to form

$$p_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ .. \\ 0 \end{pmatrix}$$

This is because since the walker is in the first cell, he is there with probability unity. After the first step, the probability vector becomes

$$p_1 = \begin{pmatrix} P(1 \longrightarrow 1) \\ P(1 \longrightarrow 2) \\ P(1 \longrightarrow 3) \\ .. \\ P(1 \longrightarrow N) \end{pmatrix}$$

Similarly, if he were to start in cell 2, the probabilities would be $P(2 \longrightarrow 1), P(2 \longrightarrow 2), .., P(2 \longrightarrow N)$ and so on. We can arrange these probablilities in the form of a matrix

$$P = \begin{pmatrix} P(1 \longrightarrow 1) & P(2 \longrightarrow 1) & .. & P(N \longrightarrow 1) \\ P(1 \longrightarrow 2) & P(2 \longrightarrow 2) & .. & P(N \longrightarrow 2) \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ P(1 \longrightarrow N) & P(2 \longrightarrow N) & .. & P(N \longrightarrow N) \end{pmatrix} \tag{3}$$

Then, obtaining p_1 from p_0 reduces to the matrix operation [1]

$$p_1 = P\ p_0$$

The successive steps simply lead to successive matrix operations by the probability matrix P. Then, given any initial state p_0, after m steps, the probability vector is

$$p_m = (P)^m\ p_0$$

Alternatively, given the state after n steps, the state after $n+1$ steps is

$$p_{n+1} = P\ p_n \tag{4}$$

It should be noted that the matrix P is independent of the state of probabilities at any given step. Further, since the total probability of going from a given cell to some cell (which could be itself or a different cell) must be equal to unity, it follows that the sum of elements of any column must be equal to 1

$$\sum_{j=1}^{N} P(i \longrightarrow j) = 1\ \forall i \tag{5}$$

Given the structure of the probability matrix, it is clear that $P(i \longrightarrow j) = P_{ji}$. Then, the probability conservation relation imposes the following constraint on the matrix P

$$\sum_{j} P_{ji} = 1 \tag{6}$$

[1] The way probability vectors and the probability matrix are defined here is a little different from the way they are introduced in traditional texts on Markov Chains, where the probability vector is a row vector and the probability matrix is the transpose of the one defined here. This stems from my bias of operating matrices on column vectors.

Given the probability distribution $\pi(\mathbf{x})$ that we wish to generate through this Markovian walk, the matrix P should be such that there should exist a steady probability vector

$$p_s = \begin{pmatrix} \pi(\mathbf{x}_1) \\ \pi(\mathbf{x}_2) \\ \pi(\mathbf{x}_3) \\ .. \\ \pi(\mathbf{x}_N) \end{pmatrix} \tag{7}$$

which should satisfy

$$P\ p_s = p_s \tag{8}$$

Here, $\mathbf{x}_i$ is the value of $\mathbf{x}$ in the i^{th} cell.

That is, eventually, the process should generate a probability vector whose entries are the probabilities we wish to sample, and this vector should be such that there is no furthur change in its entries under application of the probability matrix P. Mathematically, eqn.(8) implies that this steady vector should be an eigenvector of P with eigenvalue 1. Therefore, the problem reduces to constructing the probability matrix P which encodes the rules of the Markovian walk and which is such that the probability distribution we wish to generate is its eigenvector with unit eigenvalue [2]. The challenge is to determine the matrix P such that the equilibrium distribution vector p_s is an eigenvector with eigenvalue 1.

3 Detailed Balance

To determine this matrix, we explicitly write eqn.(8)

$$\sum_j P_{ij}\pi(\mathbf{x}_j) = \pi(\mathbf{x}_i) \tag{9}$$

Note that the distribution $\pi(\mathbf{x})$ is known, and we need to generate matrix P which satisfies the above equation. One possible solution is

$$P_{ij}\pi(\mathbf{x}_j) = P_{ji}\pi(\mathbf{x}_i) \tag{10}$$

This clearly satisfies eqn.(9), since

$$\begin{aligned} \sum_j P_{ij}\pi(\mathbf{x}_j) &= \sum_j P_{ji}\pi(\mathbf{x}_i) \\ &= \pi(\mathbf{x}_i)\sum_j P_{ji} \\ &= \pi(\mathbf{x}_i) \end{aligned} \tag{11}$$

Equation (10) is the equation for *detailed balance*. This equation states something very simple: the probabilities stay steady (equal to the equilibrium distribution $\pi(\mathbf{x})$) because the flow of probability from one state to another is equal to the flow in the other direction. This implies that given the probability distribution $\pi(\mathbf{x})$, the probability matrix which *dynamically* attains this probability distribution should be such that

$$\frac{P(i \longrightarrow j)}{P(j \longrightarrow i)} = \frac{\pi(\mathbf{x}_j)}{\pi(\mathbf{x}_i)} \tag{12}$$

This equation clearly does not uniquely determine $P(i \longrightarrow j)$.

[2] The existence and uniqueness of such a 'steady' probability vector is based on certain mathematical theorems which we assume.

4 The Metropolis Algorithm

Equation (12) does not uniquely determine $P(i \longrightarrow j)$. However, the following *ansatz* works

$$P(i \longrightarrow j) = min\left(1, \frac{\pi(\mathbf{x}_j)}{\pi(\mathbf{x}_i)}\right) \tag{13}$$

This ansatz is known as the *Metropolis Algorithm*. The way it works is as follows: say, during our random walk through the space of the possible values of the random variable $\mathbf{x}$, we are in a 'cell' where the value is $\mathbf{x}_i$. We take the next step by randomly choosing another cell, and 'proposing' a move to that cell. Say, the cell corresponds to value $\mathbf{x}_j$. If the ratio $\pi(\mathbf{x}_j)/\pi(\mathbf{x}_i) > 1$, we accept the move and step onto that cell (this is because following eqn.(13), $P(\mathbf{x}_i \longrightarrow \mathbf{x}_j) = 1$). If however $\pi(\mathbf{x}_j)/\pi(\mathbf{x}_i) < 1$, then $P(\mathbf{x}_i \longrightarrow \mathbf{x}_j) = \pi(\mathbf{x}_j)/\pi(\mathbf{x}_i)$. To realize this probability, we generate a random number between 0 and 1. If the number is less than $pi(\mathbf{x}_j)/\pi(\mathbf{x}_i)$, we accept the move and step into the new cell. Else, we stay in the original cell. During this random walk, we are often evaluating the average of some function $f(\mathbf{x})$. As we step into cell $\mathbf{x}_r$, the function picks the value $f(\mathbf{x}_r)$. To step into the next cell, as discussed, we propose a move according to the Metropolis ansatz. If the move is accepted and we step into another cell with say $\mathbf{x} = \mathbf{x}_s$, the function picks up the value $f(\mathbf{x}_s)$. However, if the move is rejected, we stay in cell $\mathbf{x}_r$ and the function f (once again) picks the value $f(\mathbf{x}_r)$. Thus, on rejection of a move, we stay in the cell we occupy and for that (rejected) move, the mean value picks a contribution from that cell itself.