Recitation 7 - Markov Jump Processes and their Simulation

1- Markov Jump Process

So far, we have been learning about Markov processes that are continuous in both their state and in time. Markov Jump Processes are continuous-time processes with a discrete state space $\mathcal{S} = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$. The number of states may be finite or infinite. At any point in time, the system occupies some state $x(t) \in \mathcal{S}$.

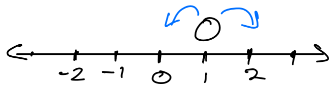
Here are some examples:

1) A chemical reaction $A \rightleftharpoons B$.

State space $\mathcal{S} = \{A, B\}$, $n = 2$.

2) A random walk on a 1D-lattice

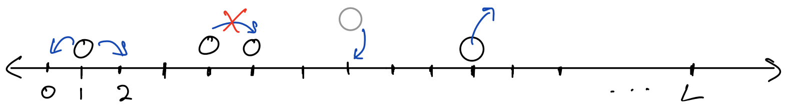
State at any time is the particle position.



Thus, $\mathcal{S} = \mathbb{Z}$, $n = \infty$

3) A collection of particles hopping on a finite 1D lattice of size L, with birth, death, and hard-core exclusion.

You can think of this as a microtubule in a solution of microtubule-binding proteins



Each state is represented by a vector of occupancy numbers

$x(t) = (n_1(t), n_2(t), \dots, n_L(t))$, where $n_i = 0$ or 1.

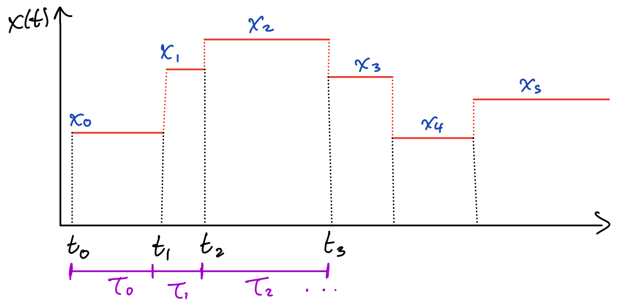
Thus $\mathcal{S} = \{0, 1\}^L$.

A trajectory of a Markov process is specified by a sequence of states $\{x_i\}$ and a corresponding sequence of dwell times:

trajectory: $((x_0, \tau_0), (x_1, \tau_1), \dots, (x_p, \tau_n), t_p)$, where $x_i \in \mathcal{S}$

The "jumps" occur at times

$t_i \equiv t_{i-1} + \tau_{i-1}$, for $i > 0$.



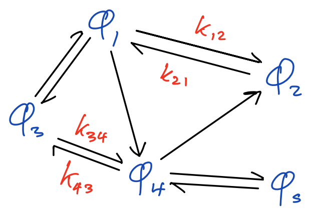
As discussed in lecture, the state occupation probabilities follow a master equation

$$\partial_t P(\varphi_i, t) = \sum_{j\neq i} W(\varphi_j \to \varphi_i) P(\varphi_j, t) - W(\varphi_i \to \varphi_j) P(\varphi_i, t)$$

where $W(\varphi_j \to \varphi_i)$ is the transition rate from state φ_j to state φ_i, defined in terms of the conditional probability

$$\lim_{\Delta t \to 0} \frac{P(\varphi_i, t+\Delta t \mid \varphi_j, t)}{\Delta t} = W(\varphi_j \to \varphi_i) \qquad \text{for } i \neq j.$$

In the context of chemical reactions, the state space is often represented as a graph, with arrows denoting allowed transitions, and the corresponding rates indicated on the arrows ($k_{ij} \equiv W(\varphi_i \to \varphi_j)$).



2- Simulating a Markov Jump Process

2.1 Dwell times and transition probabilities

To simulate a Markov jump process requires a rule for choosing the dwell time and the next state transition given the current state.

Dwell times: We showed in lecture that the dwell times are exponentially distributed according to the escape rate

$$r(\varphi_i) = \sum_{j\neq i} W(\varphi_i \to \varphi_j),$$

so that

$$\tau_i \sim \mathrm{Exp}(r(\varphi_i)), \qquad P(\tau_i) \propto r_i e^{-t}$$

Next state : We showed in lecture that, upon escaping state i, the probability that the system lands in state j is:

$$P_{i\to j} = \frac{W(\varphi_i \to \varphi_j)}{r(\varphi_i)}$$

Together, these define an algorithm to exactly sample the trajectories of a jump process, often called the Gillespie algorithm (though it is much older than Gillespie).

2.3 The Gillespie algorithm (Direct method)

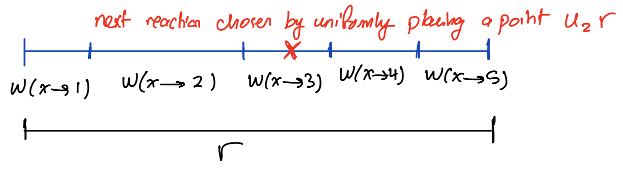
The Gillespie algorithm, developed by Doob and others in ≈ 1945 and popularized by Gillespie in 1976-77, generates the next state and jump time of a Markov jump process as follows:

1) Initialize the time and state variables $t = t_0$ and $x = x_0$.
2) Compute the escape rate $r = \sum_{\varphi \neq x} W(x \to \varphi)$
3) Draw two independent R.V.s uniformly on $[0,1]$, called u_1 and u_2.
4) Time update: set $t = t + \tau$, where $\tau \equiv \frac{1}{r} \log(1/u_1)$
5) State update: set $x = \varphi_j$, where
$$j = \min_{\ell} \left[\sum_{n=1}^{\ell} W(x \to \varphi_n) \geq u_2 r \right]$$
6) Return to step 2.

Why does this work?

$\Rightarrow$ For step (4), check that $\frac{1}{r} \log(1/u_1) \sim \mathrm{Exp}(r)$.

$\Rightarrow$ For step (5), we wish to pick the next state with a probability proportional to the rate of jumping from x to that state. See the below picture:



$\Rightarrow$ Why are the dwell time and next reaction independent? Go back to today's lecture notes and check that the joint density $p(\tau, j)$ is $p(\tau,j) = W(x\to j) \exp\left[\sum_{\ell} -W(x\to \ell)\tau\right]$

This algorithm is exact and is sometimes called the "direct" method. It is "rejection free", in the sense that every random number generation is asscociated with a state transition. It is an example of a "kinetic Monte Carlo" algorithm.

In many situations (e.g. large reaction networks, many particles on a lattice), the number of possible state transitions is very large. This means the escape rates are very large, and the dwell times are very short, so very many iterations of the algorithm are needed to elapse a reasonably long time interval. It also means that computation of the escape rates is

itself costly as it takes a time which scales linearly with the number of possible transitions.
There are some optimizations that can be done when many of the reactions are independent (e.g. far-away particles on a lattice). These rely on an alternative description of a Markov jump process.

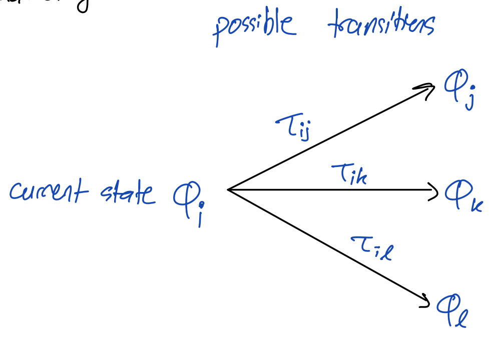
2.3 "First-reaction" and "Next reaction" methods

In the above, we characterize a Markov jump process in terms of the distributions governing the dwell time and the next state transition choice. An equivalent description is the following: suppose that the system is in state φ_i at some time. To each possible escape reaction $\varphi_i \longrightarrow \varphi_j$ is associated a stopwatch, and a waiting time

$$\tau_{ij} \sim Exp(W(\varphi_i \longrightarrow \varphi_j))$$

is drawn. The stop watches are started. The first time a stopwatch hits τ_{ij} for some j, reaction $\varphi_i \longrightarrow \varphi_j$ is executed. The waiting times are then drawn anew, and the stopwatches restarted.

Schematically:



Dwell time: $\min_j (\tau_{ij})$

Next state: φ_p, where $p = \operatorname{argmin}_j (\tau_{ij})$

To see that this is equivalent to the above, the following must be shown:

Consider a collection of n independent exponentially-distributed random variables with rates $k_1, k_2, \dots, k_n$; that is, $\{\tau_1, \dots, \tau_n\}$ with $\tau_i \sim Exp(k_i)$

(1) The minimum $\tau \equiv \min\{\tau_i\}$ is $Exp(r)$ distributed with $r = \sum_i k_i$.

(2) The probability that τ_i is the smallest among the $\{\tau_j\}$ is:

$$\mathbb{P}\left[\operatorname{argmin}_j\{\tau_j\} = i\right] = k_i / r$$

For (1), it suffices to prove the statement for $n=2$, because it may then be applied pairwise $\min(\tau_1, \tau_2, \dots, n) = \min(\min(\tau_1, \tau_2), \tau_3, \dots, \tau_n) = \min(\min(\min(\tau_1,\tau_2),\tau_3), \tau_4, \dots, \tau_n)$ and so on. For $n=2$, we have

$$\mathbb{P}(\min(\tau_1,\tau_2) > t) = \mathbb{P}(\tau_1 > t)\,\mathbb{P}(\tau_2 > t) = \int_t^\infty ds\, k_1 e^{-k_1 s} \int_t^\infty ds'\, k_2 e^{-k_2 s'}$$
$$= e^{-k_1 t - k_2 t}$$
$$= 1 - \mathbb{P}(\min(\tau_1,\tau_2) < t)$$

we thus identify the CDF as that of an $\mathrm{Exp}(k_1+k_2)$ R.V. ▨

For (2), we use indicator variables

$$\mathbb{P}\left[\operatorname*{argmin}_j \{\tau_j\} = i\right] = \left\langle \prod_{p \neq i} \mathbb{1}(\tau_k > \tau_i) \right\rangle$$

where $\mathbb{1}(A) \equiv 1$ if event A occurs and zero otherwise.

The terms within the product are not independent: if I tell you $\tau_k > \tau_i$ for some k, this tells you something about the smallness of τ_i, so that $(\tau_m > \tau_i)$ is now more likely.

To resolve this, we condition on τ_i and use the tower rule $\mathbb{E}(X) = \mathbb{E}_Y\left[\mathbb{E}_X[X|Y]\right]$:

$$\mathbb{P}\left[\operatorname*{argmin}_j \{\tau_j\} = i\right] = \left\langle \left\langle \prod_{p\neq i} \mathbb{1}(\tau_p > \tau_i) \,\middle|\, \tau_i \right\rangle \right\rangle_{\tau_i}$$
$$= \left\langle \prod_{p \neq i} \mathbb{P}(\tau_p > \tau_i) \right\rangle_{\tau_i}$$
$$= \left\langle \prod_{p\neq i} e^{-k_p \tau_i} \right\rangle_{\tau_i}$$
$$= k_i \int_0^\infty dt\, \exp\left(-\sum_{j=1}^n k_j t\right)$$
$$= \frac{k_i}{\sum_{j=1}^n k_j} \quad ▨$$

2.3.1: "First reaction" method

Based on the above Gillespie proposed the following algorithm: at each step, draw an $\mathrm{Exp}(w(x \to \varphi))$ variable τ_φ for every $\varphi \in \mathcal{S}$. Set $t \to t + \min\{\tau_\varphi\}$ and $x \longrightarrow \operatorname*{argmin}_\varphi \{\tau_\varphi\}$.

This algorithm is obviously less efficient because it draws up to $|\mathcal{S}|$ random variables at each step and identifies their minimum. Its usefulness is thus mostly pedagogical.

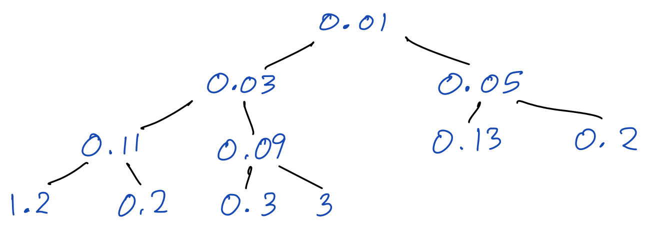
2.3.2: "Next reaction" method (Gibson and Bruck, 2000).

The idea here is to draw all the possible transition times $\{\tau_i\}$ and to store them in an efficient data structure that maintains their order.

At each step, the smallest τ_i transition is executed, and only those reactions which are affected are redrawn.

The original implementation is specialized to chemical reactions, so we list here only the main idea:

$\Rightarrow$ The transition times are stored in a min heap data structure (usually indexed)



$\Rightarrow$ The next reaction is always the root and can be found in $\mathcal{O}(1)$ time.

$\Rightarrow$ An additional data structure keeps track of the dependency structure between the elements (e.g. nearest neighbor sites on a lattice).

Example: spins on a lattice (e.g. a kinetic Ising model):

Consider a model with spins $s = \pm 1$ on a lattice $\mathbb{Z}^d$. Spin i flips at a rate which is a function of s_i and the nearest neighbors of i, denoted $\sigma(i)$:

$$\text{flip rate} = w(\{s_j : j \in \sigma(i)\}).$$

This can be simulated as follows:

1) Initialize the spin lattice. For each $i \in L^d$, draw the flip time of spin i according to $\tau_i \sim \mathrm{Exp}(w(\{s_j : j \in \sigma(i)\}))$

2) Organize the L^d pairs (i, τ_i) into a min heap sorted according to τ_i. Create a data structure of pointers to the tree for each i.

3) Set the simulation time to $t = \tau_j$, where $j = \operatorname{argmin}_i \{\tau_i\}$.

4) Flip spin j.

5) For each $i \in \sigma(j)$ and for j itself, redraw τ_i as $\tau_i \to t + \Delta\tau_i$, where $\Delta\tau_i \sim \mathrm{Exp}(w(s_k : k \in \sigma(i)))$. Re-organize the heap. Return to step 3.

The slowest step is the time to reorganize the heap, which is $\mathcal{O}(\log N)$ instead of $\mathcal{O}(N)$.
This is a big improvement over the direct method

2.4 Discrete-time (rejection) algorithms