Stochastic Systems

Not all physical laws are deterministic differential equations. Often we have to deal with stochastic systems. The source of randomness could be true random events such as nuclear decay or the measurement of a wave function, or it could be the seemingly random behaviour of stock markets or microorganism motility. Perhaps the most famous stochastic system is that of Brownian motion: the random movement of small particles due to thermal noise.

To able to simulate stochastic systems we need the computer to able to sample random numbers. For instance, the time t between events of radioactive decay is exponentially distributed

$$p(x) = \lambda e^{-\lambda t}. \tag{5.1}$$

So in order to simulate such a system, we need to be able to sample exponentially distributed numbers. Most programming languages provide random number generators for standard distributions, but custom methods are needed for special distributions. We begin this chapter with discussing this, and end with a few methods that are typically used for simulating specific random systems.

5.1 Random Numbers

To simulate random events on a computer, you need to be able to sample random numbers. However, a computer is a deterministic machine and cannot do anything truly random. All we can do are some mathematical operations that make it seem random enough. This is called pseudo-random number generation. To give a simple example, consider the sequence generated by[1]

$$x_{n+1} = 48271\, x_n \mod (2^{32} - 1). \tag{5.2}$$

[1] Recall that $a \mod b$ means the integer remainder of the division a/b.

We start with some *seed* x_0 and then keep applying the above formula. For instance, starting with $x_0 = 1656264184$ yields

$$x_1 = 3007196734, \quad x_2 = 3383877799, \quad x_3 = 1264039384 \cdots \tag{5.3}$$

In this way we can generate pseudo-random integers between 0 and $2^{32}-2$. The initial seed x_0 could be taken from some source that constantly changes such as the time on the computer in microseconds, or similar. If we need random integers between 0 and some number N, we simply use $y_n = x_n \mod (N+1)$.

In physics we are typically interested not in integers, but real numbers. If, for instance, we need random numbers sampled between 0 and 1 we could then simply take

$$r_n = \frac{x_n}{2^{32}-2}, \tag{5.4}$$

which is a good approximation to a random uniform number.

The scheme we just presented is not great though. There are much better versions, and you will probably never need to implement your own.

It is good to understand, nonetheless, the principles behind such number generation. In this chapter we are going to assume that you have access to a library that reliably generates pseudo-random numbers. In particular, we will assume that you can generate integers, both uniformly and Poisson distributed, and real numbers, both uniformly and normally distributed.

5.1.1 Inverse Transform Sampling

Suppose you need to sample random numbers from a distribution $p(x)$, but this distribution is not implemented in your language of choice. This is in fact a very common situation. If the probability distribution is simple (and 1D), the best method to use, by far, is inverse transform sampling.

Inverse Transform Sampling

To sample from a probability distribution $p(x)$, solve for the inverse cumulative distribution $Q(u)$:

$$\int_0^x p(x')\,\mathrm{d}x' = u \Leftrightarrow x = Q(u). \tag{5.5}$$

Now sample a uniform random number U between 0 and 1. The number

$$X = Q(U) \tag{5.6}$$

will then be a random number sampled from $p(x)$.

Proving this method works is quite simple, although we will skip a formal derivation here. Intuitively, nonetheless, you can note that the cumulative distribution is always a function that maps an input x to a number between $[0, 1]$. We choose a random location on this y-axis and ask which x that corresponds to by using the inverse function.

Example

To sample a random number from the exponential distribution $p(x) = \lambda e^{-\lambda x}$ (defined on $[0, \infty]$) we first need solve

$$\int_0^x \lambda e^{-\lambda x'} \, \mathrm{d}x' = 1 - e^{-\lambda x} = u \tag{5.7}$$

for x as a function of u. This one is easy and we find

$$Q(u) = -\frac{1}{\lambda} \log(1 - u). \tag{5.8}$$

Now we can use a standard sampler on a uniform interval to sample exponentially distributed numbers. Note that if U is uniform on $[0, 1]$ then so is $1 - U$, so we can also use

$$Q(u) = -\frac{1}{\lambda} \log(u). \tag{5.9}$$

Observe that indeed $Q(u)$ will map to $[0, \infty]$ for input in $[0, 1]$, as must be the case.

The formula can be used even if the equation cannot be solved analytically, as a numeric solution is adequate. In fact, not even the integral needs to be solved analytically, but helps in terms of speed of the algorithm.

5.1.2 Rejection Sampling

Inverse transform sampling works well for simple one-dimensional problems, but sometimes calculating the cumulative distribution and its inverse is a hard problem in itself. If this is the case, one can turn to rejection sampling:

Rejection Sampling

To sample from a probability distribution $p(x)$, choose a proposal distribution $q(x)$ from which it is simpler to sample and which is non-zero for all x where $p(x)$ is non-zero.
Find an M (preferably as small as possible) such that

$$p(x) \leq Mq(x) \qquad \text{for all } x \tag{5.10}$$

Then

1. Sample an x from $q(x)$
2. Sample a uniform random number U on $[0, 1]$
3. - If $U \leq \frac{p(x)}{Mq(x)}$ keep the sample
   - Otherwise start over.

This method is also very simple to use, but how fast it is depends on the choice of $q(x)$. Preferably, q should be chosen to be as close to $p(x)$ as possible in order to avoid rejection by the last step. Intuitively, you expect to sample about M numbers before getting an acceptance. Therefore it is important to choose a q that minimizes M.

We will also skip a formal derivation of this method, but again it should be fairly intuitive: You sample from q and then adjust for the fact that this is the wrong distribution by making samples more unlikely in proportion to the distance $|p(x) - M\,q(x)|$.

Example

Consider sampling from the two-dimensional distribution

$$p(x, y) = \frac{1}{4\pi^2}\left(1 + \cos(x + y)\right) \tag{5.11}$$

for $x, y \in [0, 2\pi]$. As proposal distribution we simply choose the uniform distribution

$$q(x, y) = \frac{1}{4\pi^2}, \tag{5.12}$$

which is extremely easy to sample from, as we just sample two uniform random numbers on $[0, 2\pi]$. The maximal value of $p(x, y)$ is $\frac{1}{2\pi^2}$, and so the best M we can choose is $M = 2$.

5.1.3 Markov Chain Monte Carlo

Sometimes the probability distribution you are trying to sample from is too complicated for rejection sampling to work well. In this case you can turn to Markov Chain Monte Carlo (MCMC). The idea of MCMC is to throw away the requirement that we need independent samples. Instead we create a long sequence of samples where each sample is allowed to be correlated with the previous, but have the same statistics as independent samples if shuffled.

MCMC is a random walk in parameter space, biased in such a way that we spend more time in areas of high probability. We carefully choose the biasing such that after a long time, we have perfectly sampled the probability distribution.

We present only the simplest version of Markov Chain Monte Carlo, which depend on a choice of jump distribution $g(x \mid x')$ that is easy to sample from. For this one, if the parameters are continuous, we will often choose a Gaussian

$$g(x \mid x') = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-x')^2/2\sigma^2}. \tag{5.13}$$

MCMC: Metropolis–Hastings

To sample N points from $p(x)$ choose a jump distribution $g(x \mid x')$ from which it is easy to sample. Choose a starting point x_0. Then for $n \in [1, 2, \cdots N]$

1. Sample x from $g(x \mid x_{n-1})$.
2. Calculate $\alpha = \frac{p(x)}{p(x_{n-1})} \frac{g(x_{n-1} \mid x)}{g(x \mid x_{n-1})}$
3. - If $\alpha > 1$ set $x_n = x$
   - Otherwise sample a uniform random number U on $[0, 1]$.
     - If $U \leq \alpha$ set $x_n = x$
     - Otherwise set $x_n = x_{n-1}$.

For sufficiently large N, the sequence of samples can be shuffled to emulate independent samples of p.

Note that if g is symmetric, then $g(x_{n-1} \mid x)/g(x \mid x_{n-1}) = 1$, simplifying the method. This is for instance the case for Eq. (5.13).

The early samples of the sequence will depend on the choice of x_0. Therefore it is advisable to discard the first many samples (say the first 1,000, depending on the distribution being sampled). This is called the warmup or burn-in period.

The choice of g will massively affect the efficiency of the method. In the case that you use Eq. (5.13) for $g(x)$, how would you choose σ? For high-dimensional problems, the optimal choice is one that leads to acceptance probability of about 23 %. This can be tuned during warmup.

Note that Markov Chain Monte Carlo works even if you do not have access to a normalised distribution, as it only uses $p(x)/p(x')$. This is extremely useful both for physical simulations and for data modelling.

Physical simulation — As an example of using MCMC to do physical calculation we consider the Boltzmann distribution of statistical physics

$$p(\boldsymbol{x}) = \frac{e^{-E(\boldsymbol{x})/T}}{Z}. \tag{5.14}$$

Here, $\boldsymbol{x}$ is the micro-state of the system, E is the energy of the that state, T the temperature, and Z the partition function. The partition function is simply a normalisation, but is typically very hard to calculate. To calculate statistics of such a model, we can use MCMC to sample the distribution of microstates, without having to evaluate Z. In particular, if we use a symmetric jump distribution g, we simply have $\alpha = e^{-\Delta E/T}$. When used for physical simulations like this, the method is often referred to simply as *The Monte Carlo Method*, although this strictly speaking refers to the broad range of methods that use random number generation.

Data modelling — Consider the case, where you are trying to estimate some parameters x and y based on some data. From physical principles you derive[2] $p(\text{data} \mid x, y)$. This is typically what you can get from physics: if we already knew the values of x and y, we can calculate the probability of observing the data we did. From background information we also typically have a prior on x and y: $p(x, y)$. Then from Bayes' formula we have

$$p(x, y \mid \text{data}) = \frac{p(\text{data} \mid x, y)\, p(x, y)}{p(\text{data})}, \tag{5.15}$$

which is the function you want to sample from. The normalisation $p(\text{data})$ is hard to calculate, especially for high-dimensional problems, and so we typically only have access to

$$p(x, y \mid \text{data}) \propto p(\text{data} \mid x, y)\, p(x, y) = \mathcal{L}(x, y), \tag{5.16}$$

where $\mathcal{L}(x, y)$ is called the likelihood function. Fortunately, this is enough for MCMC to be able to sample x and y from $p(x, y \mid \text{data})$. It will furthermore typically be better to work in terms of log likelihoods to minimise floating points errors. In the case of symmetric g, the formula for α then becomes

$$\alpha = \exp(\log \mathcal{L}(x) - \log \mathcal{L}(x_{n-1})) \tag{5.17}$$

Often Markov Chain Monte Carlo is used to evaluate integrals of the form

$$I = \int_{-\infty}^{\infty} f(x, y)\, p(x, y)\, \mathrm{d}x\, \mathrm{d}y. \tag{5.18}$$

[2] The notation $p(A|B)$ meaning the probability of A conditional on B having occurred.

For instance, if $f(x, y) = x$, we calculate the mean value μ_x of x, and $f(x, y) = (x - \mu_x)^2$ will give the variance. To evaluate these integrals normally we would need a normalised probability distribution, but with Markov Chain Monte Carlo, we can estimate it as

$$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i), \tag{5.19}$$

where (x_i, y_i) are the sampled values of x and y. The error on the estimation of I will be of order $O(N^{-1/2})$.

Finally, we note that Metropolis–Hastings is not the only MCMC method. More efficient methods (for continuous parameters) exploit knowledge of the gradient of $p(x)$. These are called Hamiltonian Monte Carlo methods and are beyond the scope of this text.

5.2 Event-based Simulations

The outcome of whether a coin falls heads or tails up is easy to simulate. We could simply sample one of the integers $\{0, 1\}$, and denote tails with zero and heads with one. This is perhaps the simplest form of an *event*-based simulation. If the coin is biased, e.g. tails happen with probability p, we could instead sample a uniform number U between [0, 1] and ask if $U < p$, in which case the toss is tails, and heads otherwise. This simple approach allows us to simulate event-driven systems. However, many physical systems are not formulated directly in terms of probabilities of events but instead in terms of *rates*. This section deals with how to simulate such stochastic systems.

5.2.1 Constant rates: Gillespie Algorithm

An example of a stochastic system specified in terms of rates is that of a chemical reaction such as

$$A + B \xrightarrow{k} C. \tag{5.20}$$

Here, A and B react together to produce C with rate k. If we have a large number of reactions, such system can readily be described by differential

equations[3] as fluctuations will not be important. However, if we consider the reaction of a small number of reactants, fluctuations cannot be ignored.

For Eq. (5.20) let us denote the number of A reactants at time t by $N_A(t)$. Likewise, we define $N_B(t)$ and $N_C(t)$. The instantaneous total reaction rate will be equal[4] to $k\,N_A(t)\,N_B(t)$. At time t, how long until the next reaction occurs? By the very definition of rates, the chance that an event with rate R occurs in a short time interval Δt is $R\Delta t$. So the probability distribution of the time Δt until next event is exponential:

$$p(\Delta t) = Re^{-R\Delta t}. \tag{5.21}$$

In the case of Eq. (5.20), $R(t) = k\,N_A(t)\,N_B(t)$. Finally, note that between events $R(t)$ is constant.

We now have all the ingredients to simulate an *exact* stochastic realization of Eq. (5.20). We simply sample a Δt from Eq. (5.21), update time $t \leftarrow t + \Delta t$ as well as the molecular count $N_A \leftarrow N_A - 1$, $N_B \leftarrow N_B - 1$, and $N_C \leftarrow N_C + 1$, since after a reaction there will be one less A and B molecules, and one more C. This approach is called the Gillespie algorithm.

It is only slightly more complicated to simulate a system in which many type of events can occur. In the following we use $\boldsymbol{x}(t)$ to denote the current state. For Eq. (5.20) this would be $\boldsymbol{x}(t) = (N_A(t),\, N_B(t),\, N_C(t))$.

The Gillespie Algorithm (Version 1)

Repeat until end of simulation:

1. Calculate current rates $\{r_i(t)\}$ using the current $\boldsymbol{x}(t)$.
2. For each event sample Δt_i from an exponential distribution with parameter r_i.
3. Find the event corresponding to the minimum value sampled: $j = \arg\min_i\{\Delta t_i\}$.
4. Let $t \leftarrow t + \Delta t_j$ and update $\boldsymbol{x}(t)$ according to event j.

[3] In this case we could for instance have $c'(t) = k\,a(t)\,b(t)$.

[4] We are sloppy with the definition of k here, as it should be rescaled according to the volume of the system under consideration.

If the system being considered has a large number of events that can occur, it is slightly more efficient to use a different, but equivalent implementation:

The Gillespie Algorithm (Version 2)

Repeat until end of simulation:

1. Calculate current rates $\{r_i(t)\}$ using the current $\boldsymbol{x}(t)$.
2. Calculate the total rate $R = \sum_i r_i$.
3. Sample Δ_t from an exponential distribution with parameter R.
4. Sample a uniform number U between 0 and R.
5. Find the first event j such that $\sum_{i=1}^{j} r_i \geq U$.
6. Let $t \leftarrow t + \Delta t$ and update $\boldsymbol{x}(t)$ according to event j.

This version only needs two random number samples, independent of the number of events. It is useful to know both of these methods as they are both often used. The two are equivalent because the distribution of the variable $X = \min(X_1, X_2, \cdots X_N)$ is exponential with parameter $R = r_1 + r_2 + \cdots r_N$ if X_i is exponentially distributed with parameter r_i.

The Gillespie algorithm simulates exact realizations of the stochastic rate equations. We recommend its use whenever it is possible. The only downside to the algorithm is that it becomes really slow if the rates are large, since the effective time step it takes will be of size $\Delta t \sim 1/R$.

In these cases we can turn to approximate methods, the simplest of which is called Tau-Leaping.

Tau-Leaping

Choose a time step Δt, and repeat until end of simulation:

1. Calculate current rates $\{r_i(t)\}$ using the current $\boldsymbol{x}(t)$.

2. For each event i, sample N_i from a Poisson distribution with parameter $r_i \Delta t$:

$$p(N_i) = \frac{(r_i \Delta t)^{N_i} e^{-r_i \Delta t}}{N_i!}. \tag{5.22}$$

3. Let $t \leftarrow t + \Delta t$. For each event i, update $\boldsymbol{x}(t)$ by having the event occur N_i times.

We note that the method is called τ-leaping, because Δt is usually written using τ. We prefer Δt for consistency with the other methods, however. The above scheme is approximate, as we assume $\boldsymbol{x}(t)$ constant in the time between t and $t + \Delta t$, even though many events could occur in that time.

For differential equations we described the accuracy of a numerical scheme by how the error scaled with Δt. It is slightly harder to define the error for a stochastic simulation, since each time you run a simulation a different result will be found. This is the point of stochastic simulations after all. To define an error we ask what happens if we simulate many times and compare the average of such simulations to a true realization (such as one found by the Gillepsie algorithm). We have two choices for how to define this average error: we can take the error of the means or we can take the mean of the errors.

A method is described to have a *strong* order of convergence $O(\Delta t^n)$ if[5]

$$\max_t \mathbb{E}\,|X_{\text{sim}}(t) - X_{\text{true}}(t)| = O(\Delta t^n), \tag{5.23}$$

where $\mathbb{E}$ denotes expectation (averaging over all simulations).

A method is described to have a *weak* order of convergence $O(\Delta t^n)$ if

$$\max_t |\mathbb{E}[X_{\text{sim}}(t)^m] - \mathbb{E}[X_{\text{true}}(t)^m]| = O(\Delta t^n) \tag{5.24}$$

for all integer values of m. Note that if Eq. (5.23) holds, then so does Eq. (5.24), but not the other way around.

Tau-leaping is order $O(\Delta t)$ in the weak convergence, but only $O(\sqrt{\Delta t})$ in strong convergence. You therefore need to use very small Δt when using

[5]For simplicity we write maximum. To be mathematically precise this should be a supremum.

this method. But very small can still be significantly larger than what is required by the Gillespie method for problems with large rates. Note that the definition of error is over the entire simulation, not per time step. Thus, the above should be compared e.g. to the total error of the Euler method of $O(\Delta t)$ (since for ODEs the largest error will typically be found at the last time step $t = T$).

5.2.2 Time-dependent rates

Consider a living cell that is dividing. We define it to be born at time t_0. What is the time before it makes the next division?
This is clearly an example where the rate for next event is not constant: A cell just formed does never divide right away.
As a simple model we consider the rate for division to be:

$$r(\tau) = \frac{a}{1 + e^{-b(\tau - \hat{\tau})}} \tag{5.25}$$

In this example, we define $\tau = t - t_0$ and $\hat{\tau}$ would be the typical division time. Like with the Gillespie algorithm, we can define the probability for the event to occur in the time interval between τ and $\tau + dt$ to be:

$$p(\tau) = \prod_{j=1}^{\tau/dt} \Big(1 - r(j \cdot dt)dt\Big) r(\tau)dt \tag{5.26}$$

This can be solved directly using the Volterra product integral:

$$\prod_{j=a}^{b} \Big(1 - r(t)dt\Big) = e^{-\int_a^b r(t)dt} \tag{5.27}$$

The probability that the event has not happened at time t is therefore:

$$\int_0^{\tau} p(\tau')d\tau' = 1 - e^{-\int_a^b r(t)dt} \tag{5.28}$$

Therefore one can simulate any time dependent rate numerically, by using rejection sampling based on the calculated distribution.

Example

Consider the linearly, time-dependent reaction rate:

$$r(t) = kt$$

The probability that the reaction occurs between time t and $t + dt$ is:

$$p(t, t + dt) = e^{-\frac{1}{2}kt^2} kt\mathrm{dt}$$

This probability we can simulate using rejection sampling or a combination of the Rejection sampling and Inverse Transform Sampling. Note that for all time dependent reactions, it is extremely easy to calculate the probability that a reaction has not occurred after time t:

$$p_{not}(t) = \int_0^t e^{-\frac{1}{2}kt'^2} kt'\mathrm{dt}' = 1 - e^{-\frac{1}{2}kt^2}$$

5.3 Stochastic Differential Equations

Event-based stochastic systems are discrete in time in the sense that there are finite periods of time over which nothing happens. In contrast, Stochastic Differential Equations (SDEs) are the stochastic generalisation of Ordinary Differential Equations. Here we consider SDEs of the form

$$\mathrm{d}X = \mu(X, t)\,\mathrm{d}t + \sigma(X, t)\,\mathrm{d}W. \tag{5.29}$$

If you have never seen this notation before it can be a bit weird. Informally, you have think of $\mathrm{d}W$ as an infinitesimally small random number:

$$\mathrm{d}W = \lim_{\Delta t \to 0} \Delta W, \tag{5.30}$$

where ΔW is a normally distributed random number with mean zero and variance Δt. Note that this means that the standard deviation of ΔW is $\sqrt{\Delta t}$. Physicists often use the notation

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \mu(X, t) + \sigma(X, t)\,\xi(t), \tag{5.31}$$

where $\xi(t)$ is a noise term. The former notation, however, is mathematically more well-defined and in fact also more natural for introducing numerical methods.

5.3.1 Initial-Value Problems

Just as for initial-value problems for ODEs, for SDEs we also choose a finite step size Δt. Almost as simple as the Euler Method, is the Euler–Maruyama method for SDEs:

Euler–Maruyama Method

Each time step is taken by updating

$$X(t + \Delta t) = X(t) + \mu(X(t), t)\Delta t + \sigma(X(t), t)\Delta W, \tag{5.32}$$

where ΔW is a normally distributed random number with mean zero and variance Δt:

$$p(\Delta W) = \frac{1}{\sqrt{2\pi\Delta t}} e^{-\Delta W^2/2\Delta t}. \tag{5.33}$$

The Euler–Maruyama method has weak order of error $O(\Delta t)$, but only $O(\sqrt{\Delta t})$ in strong order of convergence. When $\sigma(X, t)$ does not depend on X though, the strong order of convergence is $O(\Delta t)$.

If σ does depend on X, a slightly better method which for all choices of σ has $O(\Delta t)$ in strong order of convergence is:

Milstein method

Each time step is taken by updating

$$\begin{aligned} X(t + \Delta t) =& X(t) + \mu(X(t))\Delta t + \sigma(X(t))\Delta W \\ &+ \frac{1}{2}\sigma(X(t))\,\sigma'(X(t))\left(\Delta W^2 - \Delta t\right), \end{aligned} \tag{5.34}$$

where ΔW is a normally distributed random number with mean zero and variance Δt. Both occurrences of ΔW in Eq. (5.34) refer to the same random number.

Generalisations of Runge–Kutta methods to SDEs also exist (which do not use derivatives of σ), but these are beyond the scope of this text.

5.3.2 Space dependent stochasticity profiles

The SDE we have considered has been using the Itô integral. If you are aware of the distinction between Itô and Stratonovich integrals, you will know that Stratonovich SDEs are more common in physics than Itô. Conveniently, any Stratonovich SDE can be rewritten in the Itô interpretation by calculating the *noise-induced drift* term. After such a conversion the above methods can be applied. We note, nonetheless, that specific schemes designed for Stratonovich SDEs also exist.
Without diving too deep into the theory of stochastic differential equations (consider taking the course "Diffusive and Stochastic Processes"), we note that:

Noise-induced drift term

Consider a Brownian motion with space dependent stochastic profile $\sigma \mapsto \sigma(x)$: The drift term will take the form:

$$T_{drift} = \frac{1}{2}\sigma(x)\frac{\partial\sigma(x)}{\partial x}$$

For brownian particle where $D(x) = \frac{1}{2}\sigma(x)^2$ we directly obtain the motion:

$$dX = dt\Big(-\frac{D(X)}{k_B T}U(X) + \partial_X D(X)\Big) + \sqrt{2D(X)}dW$$

This can directly be simulated using the Euler-Maruyama (or Milstein) method and importantly this recover the correct Boltzmann distribution in steady state.

5.3.3 Boundary-Value Problems

You are far more likely to run into stochastic initial-value problems than boundary-value problems. Nothing, however, prevents boundary-value problems from being well-defined for stochastic problems.

The classical example of this is the Brownian bridge: a random walk for which $X(0)$ and $X(T)$ are known. How would you in this case simulate $X(t)$? This specific case has a simple solution: simulate $\tilde{X}(t)$ considering only the initial-value boundary condition $X(0)$. Then

$$X(t) = \tilde{X}(t) + \frac{t}{T}\left(X(T) - \tilde{X}(T)\right) \tag{5.35}$$

is an exact realization of the equation, which satisfies both boundary conditions.

Most problems, however, will not have such an elegant solution. One approach to the general problem is to write down a likelihood function of the stochastic simulation. For the Brownian bridge example, we could for instance write

$$\mathcal{L} = \prod_{n=1}^{N} p(X(n\Delta t) - X((n-1)\Delta t)), \tag{5.36}$$

where $N\Delta t = T$. Markov Chain Monte Carlo methods, as presented in Sec. 5.1.3, can then be used to sample for $X(\Delta t)$, $X(2\Delta t) \cdots X((N-1)\Delta t)$. Note that if we instead work in terms of log likelihood, as recommended in Sec. 5.1.3, the product in Eq. (5.36) becomes a sum.