Introduction to Uncertainty Quantification
Module 3.3: Monte Carlo Methods for Uncertainty Propagation

1 Introduction

For most practical problems, it is impossible to propagate uncertainty using exact methods that allow, for example, to perform a change of variables. Moreover, approximate methods based on Taylor series expansions are only accurate for problems that are nearly linear and/or have small uncertainties. For general problems, the most robust method for uncertainty propagation is the *Monte Carlo method* or *Monte Carlo simulation*. Numerous books have been written on Monte Carlo simulation and there is an extensive body of literature spanning many decades that treat the topic in great depth. We will not attempt to treat the method in great depth here. Rather, the intention here is to introduce the method that has become the standard for uncertainty propagation, against which all other methods are compared. We will briefly explore its foundations to understand why it works, learn how to use it, explore some of its basic properties, and learn some basic strategies to improve its convergence.

Monte Carlo simulation, and all Monte Carlo methods in general, exploit randomness to solve problems that may be deterministic. This is most common for solving problems that involve integration over some domain using random samples on the domain. Regardless of their specific implementation, all Monte Carlo methods generally perform the following steps:

1. Generate random samples on the domain
2. Perform deterministic calculations at each of the random samples
3. Statistically analyze the results

These are broad and generic steps, and may differ greatly depending on the objective and the specific method being employed. In the subsequent sections, we will explore the basics that will make these steps useful for practical problems.

2 Law of Large Numbers

The foundational principle upon which Monte Carlo simulation rests is known as the *Law of Large Numbers*. Informally, the Law of Large Numbers states that the average from the results of repeated experiments with random input will converge to the expected value as the number of experiments grows larger.

Formally, there are two versions of the Law of Large Numbers – the *Strong Law* and the *Weak Law*. These different laws relate to the convergence properties of the average toward the expected value, but are beyond the scope of our study. Instead, we will introduce the Law of Large Numbers with the caveat that different modes of convergence may be considered.

Let $X_1, X_2, \ldots, X_N$ be a sequence of independent and identically distributed (*iid*) random variables having mean value $\mathbb{E}[X_1], \mathbb{E}[X_2], \ldots, \mathbb{E}[X_N] = \mu$, the *Law of Large Numbers* states that the average:

$$\bar{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{1}$$

converges to μ as $N \to \infty$. That is:

$$\bar{X}_N \to \mu \text{ as } N \to \infty \tag{2}$$

The rate of convergence can be determined by computing the variance of the average value. If the random variables $X_1, X_2, \ldots, X_N$ have finite variance $\text{Var}(X_i) = \sigma^2, \forall i$, then

$$\text{Var}(\bar{X}_N) = \text{Var}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) = \frac{1}{N^2}\text{Var}\left(\sum_{i=1}^{N} X_i\right) = \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}. \tag{3}$$

As we will see, while the Law of Large Numbers ensures convergence, the rate of convergence in the variance is proportional to $\frac{1}{N}$ where N is the number of random samples, which is quite slow. In other words, we will need a large number of sample to ensure sufficiently small variance.

3 Monte Carlo Simulation

Consider the random variable X having probability density function $f_X(x)$. We previously defined the expectation, or expected value, of X as:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx \tag{4}$$

Thanks to the *Law of Large Numbers*, we know that if we can generate a large number of realizations of X, i.e. $X_1, X_2, \ldots, X_N$, then we can estimate $\mathbb{E}[X]$ as the average of these realizations (see Eq. (1)). That is:

$$\mathbb{E}[X] \approx \frac{1}{N}\sum_{i=1}^{N} X_i \tag{5}$$

The process of simulating a large number of realizations and averaging is known as *Monte Carlo simulation*. This, however, will require us to simulate realizations, or generate samples of X.

For the moment, let us take for granted that our computer programming language (whatever language that may be) has inbuilt functions for generating samples from a known distribution. In that case, Monte Carlo simulation is straightforward. But, the real power of Monte Carlo simulation is displayed when we consider functions of random variables. Consider the case where we have a function $Y = g(X)$ and we are interested in computing $\mathbb{E}[Y]$. We have previously seen that, if we know the distribution of X, $f_X(x)$, then in certain simple cases we can determine the distribution of Y, $f_Y(y)$, using the *Change of Variables Theorem*. However, let's now consider cases where the function $Y = g(X)$ is not available in an analytical form – perhaps $g(X)$ represents a computer model of a complicated system.

Recall that the expected value of a function of a random variable, $Y = g(X)$ can be expressed as:

$$\mathbb{E}[Y] = \mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx. \tag{6}$$

From this expectation, we can immediately see that the Law of Large Numbers can be applied to estimate the expectation by generating a large number of samples of Y, which in turn can be generated by drawing

a large number of samples of X from the known distribution $f_X(x)$ and evaluating $g(X)$ at each of these samples. Hence, the Law of Large Numbers tells us that:

$$\mathbb{E}[Y] \approx \frac{1}{N}\sum_{i=1}^{N} Y_i = \frac{1}{N}\sum_{i=1}^{N} g(X_i) \tag{7}$$

This expression can now we applied to any mathematical operator $g(X)$ and consequently, if we can draw random samples of X then we can perform Monte Carlo simulation by simply applying Eq. (7).

This process can easily be extended to a function of multiple random variables. Consider the function $Y = g(\boldsymbol{X})$ where $\boldsymbol{X} = \{X_1, X_2, \ldots, X_n\}$ is a random vector having joint probability density function $f_{\boldsymbol{X}}(\boldsymbol{x})$. By drawing random samples of $\boldsymbol{X}$ from $f_{\boldsymbol{X}}(\boldsymbol{x})$, we can once again apply the Law of Large Numbers to estimate:

$$\mathbb{E}[Y] \approx \frac{1}{N}\sum_{i=1}^{N} Y_i = \frac{1}{N}\sum_{i=1}^{N} g(\boldsymbol{X}_i) \tag{8}$$

To summarize, for any function $Y = g(\boldsymbol{X})$ (or model) having random variables inputs with known probability distribution we conduct Monte Carlo simulation through the following steps:

1. Generate N samples of the random vector $\boldsymbol{X}$ from the known joint probability density function $f_{\boldsymbol{X}}(\boldsymbol{x})$, denoted $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$.
2. Evaluation the function/model at each sample, i.e. $Y_1 = g(\boldsymbol{X_1}), Y_2 = g(\boldsymbol{X_2}), \ldots, Y_N = g(\boldsymbol{X_N})$.
3. Apply the Law of Large Numbers to estimate $\mathbb{E}[Y]$ by Eq. (8).

4 Simulating Random Variables

To perform Monte Carlo simulations, we must be able to generate random samples from a given probability distribution. Of course, we know that computers are not capable of performing random operations; computers follow deterministic algorithms that *always* lead to predictable results. Therefore, to perform Monte Carlo simulations on a computer, we must devise *pseudo-random* algorithms – that is, algorithms that generate sequences that *appear* random when subjected to statistical testing. There are numerous approaches to pseudo-random number generation and we will not elaborate further on this. Instead, we will assume that the computer is capable of generating numbers that appear random in the interval $[0, 1]$. That is, we assume that the computer can generate uniform random variables on the unit interval. Indeed, functions for generating uniform random numbers are available in all modern programming languages. This serves as the basis for all of the methods that we will discuss here.

4.1 Inverse Transform Method

Consider that we want to draw random samples of a random variable X having CDF, $F_X(x)$. Regardless of the distribution, we know that any variable defined by $U = F_X(X)$ will follow a uniform distribution on the unit interval, i.e. $U \sim U(0, 1)$. The *Inverse Transform Method* takes advantage of this relationship, specifically leveraging it's inverse operation through the *quantile function* $Q(u) = F_X^{-1}(u)$, and allows us to generate these samples in two straightforward steps:

1. Generate a sample, u, of the uniform random variable $U \sim U(0, 1)$
2. Compute $x = Q(u) = F_X^{-1}(u)$

This method will always return samples following $F_X(x)$ because $F_X(x)$ is a monotonically increasing function with $0 \leq F_X(x) \leq 1$. Since $0 \leq U \leq 1$, $F^{-1}(U)$ will always exist. The method is illustrated in Figure 1.

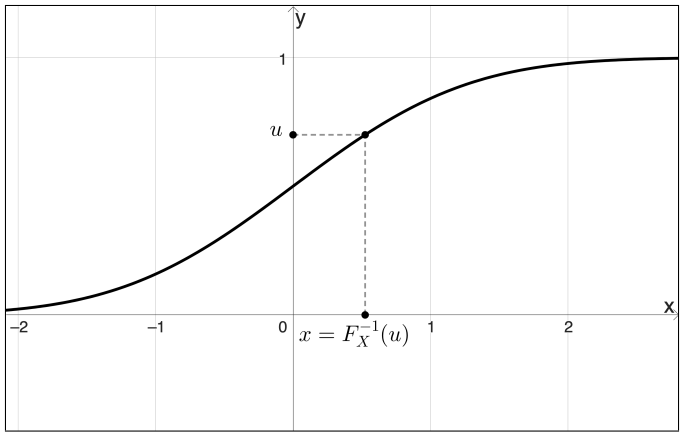


Figure 1: The inverse transform method applied to the CDF of the standard normal distribution.

The inverse transform method can be used to generate random samples from discrete distributions as well. Consider the discrete random variable X having CDF

$$F_X(x) = P(X \leq x) = \sum_{x_i \leq x} p(x_i) \tag{9}$$

where $p(x_i) = P(X = x_i)$ is the probability mass function. We apply the inverse transform through the following steps:

1. Generate a sample, u, of the uniform random variable $U \sim U(0, 1)$.
2. Find the smallest positive integer I such that $u \leq F_X(x_I)$ and return the sample x_I.

For use in Monte Carlo Simulation, the Inverse Transform Method has certain advantages and disadvantages:

Advantages

- It requires only the generation of a single uniform random variable, U, for each sample of X. Other methods, such as acceptance-rejection sampling, may require the generation of many U values for the generation of a single sample of X.
- It makes generation of random variables from truncated distributions easier by avoiding acceptance-rejection sampling. To draw samples from a truncated distribution on the range $(a, b]$, we instead simple draw uniform random samples from the range $(F_X(a), F_X(b)]$ rather than drawing them from the unit interval $[0, 1]$ again then apply $F_X^{-1}(U)$ [1].
- As we'll see later, it facilitates variance reduction techniques that rely on inducing correlation between samples.

Disadvantages

- For many distributions, such as the normal distribution (discussed next) and the Gamma distribution, the quantile function $Q(u) = F_X^{-1}(u)$ may not have an analytical form. This may require the use of numerical or expansion methods to invert the CDF, which may come at considerable expense or potentially sacrifice accuracy. Alternatively, one can develop "lookup tables" that tabulate values of $U = F_X(x)$ by computing the (forward) CDF and then interpolating between the known values.

4.2 Box-Muller Transform

As mentioned, the normal distribution does not have an analytical quantile function (inverse CDF). Therefore, the Inverse Transform Method is not commonly used to generate samples from the normal distribution. Instead, the samples from the normal distribution are often generated using the *Box-Muller Transform*.

Here we start with two uniform random variables $U_1 \sim U(0,1)$ and $U_2 \sim U(0,1)$. We then compute:

$$\begin{aligned} Z_0 &= R\cos(\Theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2) \\ Z_1 &= R\sin(\Theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2). \end{aligned} \tag{10}$$

where Z_0 and Z_1 are standard normal random variables.

This works by considering two standard normal random variables, X and Y. Under a coordinate transformation from Cartesian to polar coordinates, we see that the resulting radial and angular coordinates can be expressed as:

$$\begin{aligned} R^2 &= X^2 + Y^2 \\ \Theta &= \text{atan2}(Y, X) \end{aligned} \tag{11}$$

Because R^2 follows an exponential distribution (chi-squared distribution with two degrees of freedom), it can be expressed as:

$$R^2 = -2\ln U_1 \tag{12}$$

Meanwhile, $\text{atan2}(Y, X)$ is the angle between the positive X-axis and the vector from the origin to the point (X, Y), which can be shown to be uniform on $[-\pi, \pi]$. As a result, we can express

$$\theta = 2\pi U_2 \tag{13}$$

This can be easily proven using the *Change of Variables Theorem* we explored in previous lessons.

Therefore, to simulate normal random variables using the Box-Muller Transform, we follow two simple steps:

1. Generate two samples, u_1 and u_2, as independent uniform random variables $U_1 \sim U(0,1)$, $U_2 \sim U(0,1)$.
2. Compute z_0 and z_1 as samples of the standard normal random variables Z_0 and Z_1 using Eq. (10).

The Box-Muller Transform is illustrated in Figure 2.

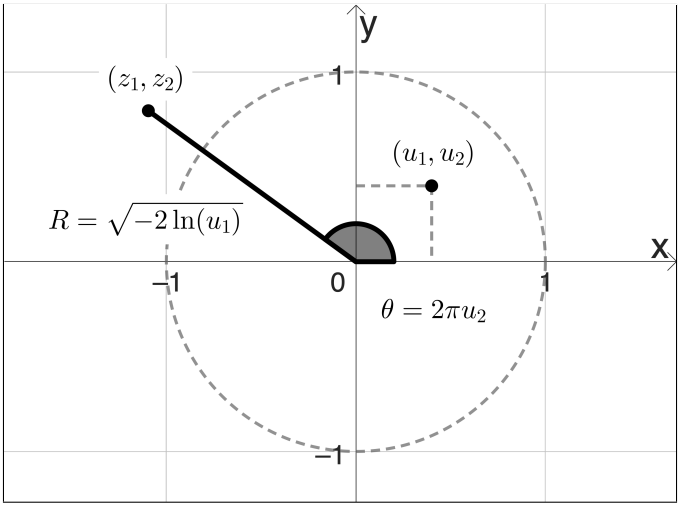


Figure 2: A graphical representation of the Box-Muller transform.

4.3 Composition Method

The *Composition Method* is a method for generating samples from a distribution that can be expressed as a convex combination of other distributions. Consider the random variable X having CDF $F_X(x)$ that can be expressed as:

$$F_X(x) = \sum_{j=1}^{N} p_j F_j(x) \tag{14}$$

where $p_j \geq 0$ and $\sum_{j=1}^{N} p_j = 1$ are *weights* attributed to each of the CDFs $F_j(x), j = 1, \ldots, N$. In similar fashion, the PDF of X can be expressed as:

$$f_X(x) = \sum_{j=1}^{N} p_j f_j(x) \tag{15}$$

To generate random variables using the *Composition Method*, we will simply generate samples from each of the distributions $F_j(x)$, each with probability p_j. This can be done in two simple steps:

1. Generate a positive random integer J with probability mass $P(J = j) = p_j, \forall j$.
2. Return a value X following distribution $F_j(x)$.

The first step of this procedure can be conducted by applying the inverse transform method to sample from the discrete probability distribution having probability mass $P(J = j) = p_j, \forall j$. The second step can be conducted using whatever sampling method is appropriate to sample from $F_j(x)$.

Example
Consider the right trapezoidal distribution having probability density function:

$$f_X(x) = \begin{cases} a + 2(1-a)x, & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

and illustrated in Figure 3.

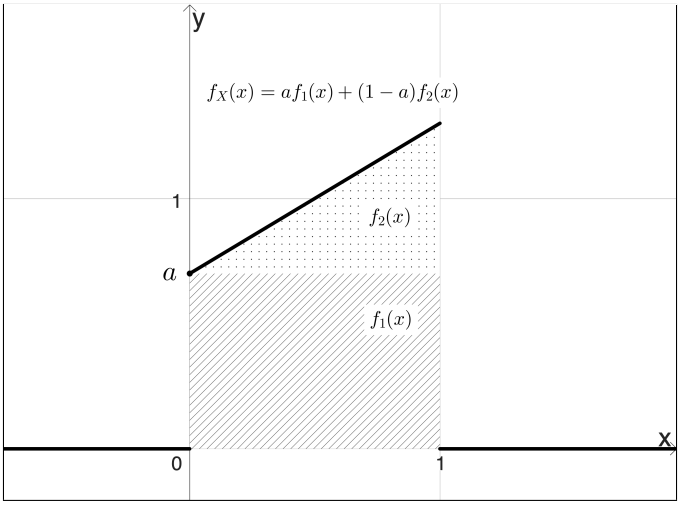


Figure 3: The distribution $f_X(x)$. Note that $f_X(x)$ can be expressed as a convex combinations of the shaded regions $f_{X_1}(x) = 1$ and $f_{X_2}(x) = 2x$.

This distribution can be expressed as the following convex combination of distributions:

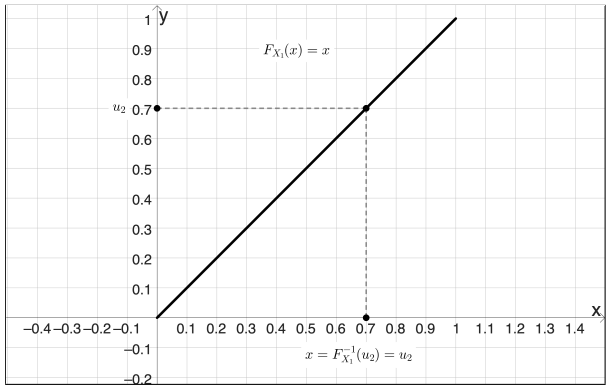
$$f_X(x) = af_1(x) + (1-a)f_2(x) \tag{17}$$

where $f_1(x) = 1, 0 \leq x \leq 1$ and $f_1(x) = 0$, otherwise (i.e. uniform distribution over $[0, 1]$) and $f_2(x) = 2x, 0 \leq x \leq 1$ and $f_1(x) = 0$, (i.e. right triangular distribution) having weights $p_1 = a$ and $p_2 = 1 - a$.

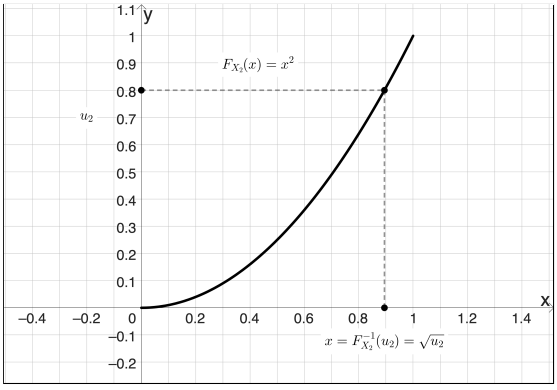
To generate a sample x from this distribution, we do the following:

1. Generate a sample u_1 following $U_1 \sim U(0, 1)$
2. If $u_1 \leq a$, generate u_2 as an independent uniform random variable $U_2 \sim U(0, 1)$ and set $x = u_2$.
3. If $u_1 > a$, apply the inverse transform by generating u_2 as an independent uniform random variable $U_2 \sim U(0, 1)$ and set $x = \sqrt{u_2}$.

A set of sample illustrating this approach are shown in Figure 4.



(a) Case $u_1 < a$: The sample x is drawn according to a uniform distribution, $x = u_2$.



(b) Case $u_1 \geq a$: The sample x is drawn using the inverse transform method, $x = F_{X_2}^{-1}(u_2) = \sqrt{u_2}$

Figure 4: Illustration of the composition method.

4.4 Convolution Method

For some important distributions, the random variable X can be expressed as the sum of other *iid* random variables $Y_1, Y_2, \ldots, Y_m$ having known distributions $f_Y(y)$. We can therefore generate samples of X by generating samples of $Y_i, i = 1, \ldots, m$ and summing as follows:

1. Generate samples y_i of each random variable Y_i according to distribution $f_Y(y)$ for all $i = 1, \ldots, m$.
2. Set $x = \sum_{i=1}^{m} y_i$.

Note the important difference here between the composition method and the convolution method. The composition method considers the distribution to be a convex combination of a set of other distributions. The convolution method, on the other hand, considers the random variable itself to be a sum other other independent random variables. We note that the latter sum of random variables results in a *convolution* of the distributions, as opposed to a summation of the distributions. This was seen in the prior section on functions of multiple random variables.

Example
The Erlang distribution is defined as the sum of m *iid* exponential random variables. Consider $Y_1, Y_2, \ldots, Y_m$ as independent random variables having exponential distribution with rate λ (i.e. mean $\mu = 1/\lambda$), then $X = \sum_{i=1}^{m} Y_i$ has Erlang distribution with PDF:

$$f(x; m, \lambda) = \frac{\lambda^m x^{m-1} e^{-\lambda x}}{(m-1)!} \quad \text{for } x, \lambda \geq 0, \tag{18}$$

4.5 Acceptance-Rejection Method

The *acceptance-rejection* method differs from the previous methods in that it does not directly sample from the given distribution, using e.g. an isoprobabilistic transformation (inverse transform method), or combinations of other random variables (composition and convolution methods). Instead, the acceptance-rejection method samples indirectly by first proposing a candidate sample and then using probabilistic criteria derived from the probability distribution of the random variable to accept or reject the sample.

Consider that we aim to draw samples of a random variable X having probability density $f_X(x)$. Let us begin by defining a function, $t(x)$, called the *majorizing function* such that $t(x) \geq f_X(x), \forall x$. Note that

$t(x)$ is *not* a probability density function in general because

$$c = \int_{-\infty}^{\infty} t(x)dx \geq \int_{-\infty}^{\infty} f_X(x)dx = 1 \tag{19}$$

However, the function $r(x) = t(x)/c$ is a valid probability density function. Our aim in the acceptance-rejection method is to design $t(x)$ such that $|t(x) - f_X(x)|$ is minimized and we can easily generate samples from $r(x)$.

The acceptance-rejection method follows three steps:

1. Generate a candidate sample, y, of random variable Y having probability density function $r(y)$. This can be done using any of the standard methods presented above.
2. Generate a sample, u, of a uniform random variable $U(0, 1)$.
3. If $u \leq \dfrac{f_X(y)}{t(y)}$, then accept the candidate y as a sample of random variable X, i.e. set $x = y$.
4. If $u > \dfrac{f_X(y)}{t(y)}$, reject the candidate y and try again.

A formal proof that the samples x_i generated using this method follow the distribution $f_X(x)$ can be found in [1]. However, an intuitive but less formal explanation follows.

Let us begin by restating the condition in Step 3 as $u \cdot t(y) \leq f_X(y)$. Next, if we plot the pair $(y, u \cdot t(y))$, then we will accept the point if the point lies below the PDF $f_X(y)$ and reject it if it lies above the curve. This is illustrated in Figure 5 where accepted points are shown with 'o's and rejected candidates are shown with 'x's.

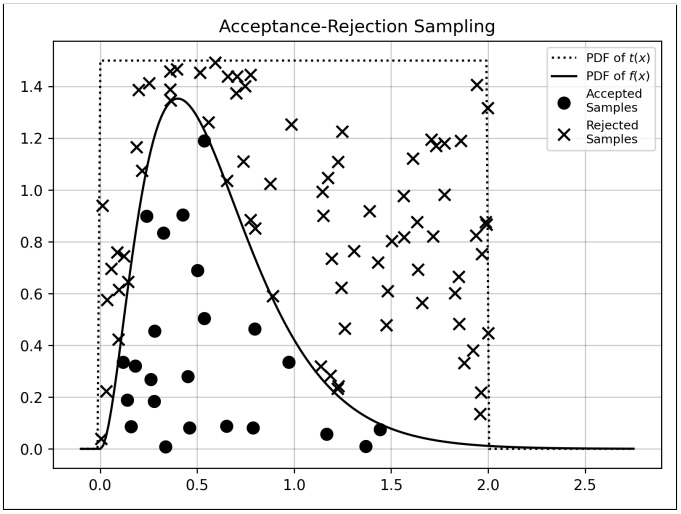


Figure 5: Illustration of the acceptance-rejection method.

The acceptance-rejection method can be very efficient or it can be very inefficient, and its efficiency depends entirely on the selection of the majorizing function $t(x)$. Majorizing functions that differ greatly from $f_X(x)$ will have very high rejection rate and will therefore be inefficient. In fact, the efficiency of the method depends on the value of c in Eq. (19) such that the probability of acceptance of a given sample is equal to

$$p_{\text{accept}} = \frac{1}{c} \tag{20}$$

and the probability of rejection is equal to

$$p_{\text{reject}} = 1 - p_{\text{accept}} = \frac{c-1}{c} \tag{21}$$

5 Simulating Random Vectors & Correlated Random Variables

Here, we now aim to generate realizations of a random vector $\boldsymbol{X} = X_1, X_2, \ldots, X_d$ having, in general, a prescribed joint distribution. First, consider that we know the complete joint CDF, given by $F_{\boldsymbol{X}}(\boldsymbol{x}) = F_{X_1 X_2 \ldots X_d}(x_1, x_2, \ldots, x_d)$. Let us further assume that we can obtain the conditional distributions $F_{X_i}(x_i|x_1, x_2, \ldots, x_{i-1})$ and the marginal distributions $F_{X_i}(x_i), , i = 1, \ldots, d$. With this complete description of the distribution, we can simulate $\boldsymbol{X}$ as follows:

1. Generate X_1 according to $F_{X_1}(x_1)$ using, e.g., one of the methods described in the previous section.
2. Generate X_2 according to the conditional distribution of X_2 given the sample x_1 generated in the previous step, i.e. $F_{X_2}(x_2|x_1)$.
3. Generate X_3 according to the conditional distribution of X_3 given the samples x_1, x_2 generated in the previous step, i.e. $F_{X_3}(x_3|x_1, x_2)$.
4. Continue this process for all components of the random vector, concluding by generating X_d from the conditional distribution $F_{X_d}(x_d|x_1, \ldots, x_{d-1})$.

The resulting components are then assembled into the random vector $\boldsymbol{X}$.

Unfortunately, this method is rarely practical because: (a) The complete joint distributions are rarely available, especially as d grows large; and (b) Even when joint distributions are available, calculation of the conditional and marginal distributions can be very difficult, if not impossible.

5.1 Simulating Normal Random Vectors

One special case where the full joint probability distribution may be known is jointly Gaussian random variables. Recall that the joint Gaussian distribution is completely defined by its first and second moments – the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{C}$. By definition, $\boldsymbol{C}$ is a symmetric and positive definite matrix. Therefore, it can be decomposed as:

$$\boldsymbol{C} = \boldsymbol{H}^T \boldsymbol{H} \tag{22}$$

There are several ways to perform this decomposition.

Cholesky Decomposition

Applying the *Cholesky decomposition*, $\boldsymbol{H}$ is a $d \times d$ upper triangular matrix having components $h_{ij} > 0, j \geq i$ and $h_{ij} = 0, j < i$. Using the Cholesky decomposed form of the covariance, we can simulate the Gaussian random vector as follows

1. Generate $\boldsymbol{Z} = [Z_1, Z_2, \ldots, Z_d]^T$ by generating each Z_i as *iid* standard normal random variables.
2. Transform these standard normal random variables by

$$X_i = \mu_i + \sum_{j=1}^{i} h_{ji} Z_j \tag{23}$$

In matrix form, this can be expressed as

$$\boldsymbol{X} = \boldsymbol{\mu} + \boldsymbol{H}^T \boldsymbol{Z} \tag{24}$$

As a simple example, consider a two-dimensional random vector having covariance given by

$$\boldsymbol{C} = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}.$$

The Cholesky decomposition yields

$$\boldsymbol{H} = \begin{bmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{bmatrix}.$$

Hence, the random vector $\boldsymbol{X} = [X_1, X_2]^T$ is computed as:

$$\begin{aligned} X_1 &= Z_1 \\ X_2 &= \rho Z_1 + \sqrt{1-\rho^2} Z_2 \end{aligned}$$

Eigen Decomposition

5.2 Simulating Correlated Non-Gaussian Random Vectors

6 Simulating Random Processes and Random Fields

7 Markov Chain Monte Carlo Methods

In many cases, it may be difficult to directly generate samples of the random vector $\boldsymbol{X}$ from its joint probability density function. This is especially true when the distribution has complex, nonlinear dependencies between its components or when the distribution is only known up to an integration constant, i.e.

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = Cp(\boldsymbol{x}). \tag{25}$$

As we'll see later, this arises frequently in Bayesian inference problems where the posterior distribution is proportional to the likelihood times the prior, but the proportionality constant is difficult to estimate.

In cases such as these, it is common to construct a Markov chain having stationary distribution $\pi(\boldsymbol{x}) = f_{\boldsymbol{X}}(\boldsymbol{x})$. If this can be achieved, it can be shown that the Law of Large Numbers remains valid and the samples of the Markov chain can be used for Monte Carlo simulations. The various methods of constructing Markov chains with the desired stationary distribution are referred to as *Markov Chain Monte Carlo* (MCMC) methods. For a review of Markov chains and the important concepts of stationarity and reversibility that will be used throughout this section, see Module 2.3. Here, we will describe several variations of MCMC starting with the most commonly used and well-known random walk Metropolis-Hastings algorithm and building from it. We will then introduce methods that use deterministic gradient-based steps to better explore the stationary distribution, take larger steps, and potentially yield more statistically representative sample sets.

7.1 Random Walk Methods

The basic premise of all random walk MCMC methods is the following. Given the current state of the Markov chain, we propose a random step (or a random transition to a new state). We then accept or reject this new step using some criterion that ensures both the stationarity and reversibility conditions,

thus ensuring that it draws samples from the correct stationary distribution. The means by which the random steps are taken and accepted or rejected differ with each algorithm. Here we will introduce several of the most commonly used random walk MCMC methods, most of which are variants of the original Metropolis-Hastings algorith, discussed first.

7.1.1 Metropolis-Hasting Algorithm

Consider an arbitrary state of a Markov chain, $\boldsymbol{x}_i$. We wish to generate samples according to the (possibly unnormalized) probability density $p(x)$. The Metropolis-Hastings algorithm works as follows:

1. Define an arbitrary transition probability density, often referred to as the *proposal density*, $q(\boldsymbol{x}|\boldsymbol{x}_i)$. Notice that we define this proposal density as the conditional density of state $\boldsymbol{x}$ given the current state $\boldsymbol{x}_i$.

2. Generate a new candidate state $\boldsymbol{x}^*$ according to $q(\boldsymbol{x}|\boldsymbol{x}_i)$.

3. Compute the product $r(\boldsymbol{x}^*, \boldsymbol{x}_i) = a_1(\boldsymbol{x}^*, \boldsymbol{x}_i)a_2(\boldsymbol{x}^*, \boldsymbol{x}_i)$ where a_1 and a_2 are defined by the following ratios:
$$a_1(\boldsymbol{x}^*, \boldsymbol{x}_i) = \frac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x_i})} \tag{26}$$
is the ratio of the probability density at the newly proposed state and the previous state, and
$$a_2(\boldsymbol{x}^*, \boldsymbol{x}_i) = \frac{q(\boldsymbol{x}_i|\boldsymbol{x}^*)}{q(\boldsymbol{x}^*|\boldsymbol{x}_i)} \tag{27}$$
is the ratio of the proposal density in the reverse direction to the proposal density in the forward direction.

4. Accept the new state with probability
$$\alpha(\boldsymbol{x}^*|\boldsymbol{x}) = \min\left(1, r(\boldsymbol{x}^*, \boldsymbol{x}_i)\right) = \min\left(1, \frac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x_i})}\frac{q(\boldsymbol{x}_i|\boldsymbol{x}^*)}{q(\boldsymbol{x}^*|\boldsymbol{x}_i)}\right) \tag{28}$$
In this final step, we always accept the proposed new state, $\boldsymbol{x}_{i+1} = \boldsymbol{x}^*$, if $r \geq 1$ and we accept it with probability r if $r < 1$. Otherwise, we reject the new state and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$.

The above steps are repeated for a large number of iterations, at which point the set of generated states will converge in distribution to $p(\boldsymbol{x})$.

The choice of acceptance density in Eq. (28) results in a reversible Markov chain having stationary distribution following $p(\boldsymbol{x})$. Recall that a Markov chain is reversible with stationary probability $p(\boldsymbol{x})$ if
$$p(\boldsymbol{x}_i)p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i) = p(\boldsymbol{x}_{i+1})p(\boldsymbol{x}_i|\boldsymbol{x}_{i+1}), \tag{29}$$
which can be rewritten as:
$$\frac{p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)}{p(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})} = \frac{p(\boldsymbol{x}_{i+1})}{p(\boldsymbol{x}_i)} \tag{30}$$
Next, recall that we separate the transition probabilities into a proposal density and an acceptance density as
$$p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i) = q(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)\alpha(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i). \tag{31}$$
We then apply this transition probability to Eq. (30) to obtain
$$\frac{\alpha(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)}{\alpha(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})} = \frac{p(\boldsymbol{x}_{i+1})q(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})}{p(\boldsymbol{x}_i)q(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)} \tag{32}$$

Applying the acceptance density in Eq. (28), we can see that the reversibility condition in Eq. (32) is satisfied for both $r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) \geq 1$ and $r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) < 1$. This is seen because, when $r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) \geq 1$, we have $\alpha(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) = 1$ and $\alpha(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) = 1/r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i)$. Furthermore, when $r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) < 1$, we have $\alpha(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i) = r(\boldsymbol{x}_{i+1}, \boldsymbol{x}_i)$ and $\alpha(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) = 1$.

Lastly, we consider the special case where the proposal density is symmetric. When $q(\boldsymbol{x}^*|\boldsymbol{x}_i) = q(\boldsymbol{x}_i|\boldsymbol{x}^*)$, we see that the ratio $a_2 = 1$ and the acceptance probability simplifies to consider only the ratio of the probability density $p(\boldsymbol{x})$ in the proposed and previous states as:

$$\alpha(\boldsymbol{x}^*|\boldsymbol{x}) = \min\left(1, \frac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x_i})}\right) \tag{33}$$

7.1.2 Gibbs Sampler

In certain cases, it may be difficult to directly sample from the complete joint distribution $p(\boldsymbol{x})$, but it may be simpler to draw samples from the conditional distributions $p(x_j|\boldsymbol{x}_{\sim j})$ where x_j is the j^{th} component of $\boldsymbol{x}$ and $\boldsymbol{x}_{\sim j}$ is the vector composed of all components of $\boldsymbol{x}$ except component j. In this case, we use the conditional distributions as the proposal distribution to sample in a component-wise manner. Consider the current state of a Markov chain given by the d-dimensional vector $\boldsymbol{x} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^T$. The Gibbs sampler then proceeds as follows:

1. Generate the first component of the new state using the proposal density
$$q(x_1^{(i+1)}) = p(x_1|x_2^{(i)}, \ldots, x_d^{(i)}) \tag{34}$$
2. Generate the second component of the new state using the proposal density
$$q(x_2^{(i+1)}) = p(x_2|x_1^{(i+1)}, x_3^{(i)} \ldots, x_d^{(i)}) \tag{35}$$
3. Continue sequentially generating each component such that the j^{th} component is generated from the proposal density
$$q(x_j^{(i+1)}) = p(x_j|x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)} \ldots, x_d^{(i)}) \tag{36}$$
4. Continue until the final component is generated from the proposal density
$$q(x_d^{(i+1)}) = p(x_d|x_1^{(i+1)}, \ldots, x_{d-1}^{(i+1)}) \tag{37}$$

Once we have iterated through the entire vector, we see that we draw the newly proposed sample by arriving at a sequence of intermediate states drawn from the conditional distributions on each sample.

The Gibbs sampler can be seen as a special case of the Metropolis Hastings algorithm. Here, at each iteration, let us consider that we define the full joint probability density $p(\boldsymbol{x})$ through the product of its marginal density for component x_j and the corresponding conditional density as

$$p(\boldsymbol{x}) = p(\boldsymbol{x}_{\sim j})p(x_j|\boldsymbol{x}_{\sim j}) \tag{38}$$

We now consider that the proposal density for component j is given by

$$q(x_j^*) = p(x_j^*|\boldsymbol{x}_{\sim j}) \tag{39}$$

Applying these relations to the Metropolis-Hastings acceptance-rejection criterion, we can see that

$$r(x_j^*, \boldsymbol{x}_{\sim j}) = \frac{p(\boldsymbol{x}_{\sim j})p(x_j^*|\boldsymbol{x}_{\sim j})p(x_j|\boldsymbol{x}_{\sim j})}{p(\boldsymbol{x}_{\sim j})p(x_j|\boldsymbol{x}_{\sim j})p(x_j^*|\boldsymbol{x}_{\sim j})} = 1 \tag{40}$$

Hence, the Gibbs sampler can be seen as a Metropolis-Hastings algorithm on each component where the proposal density yields states that are always accepted.

7.1.3 Component-wise Metropolis-Hastings

In a similar manner to the Gibbs sampler, we can sequentially update each component of the state using a method referred to as the component-wise Metropolis-Hastings algorithm (also referred to as the variable-at-a-time Metropolis Hastings and sometimes the modified Metropolis-Hastings). The difference between the component-wise Metropolis Hastings algorithm and the Gibbs sampler is that the proposal density is defined arbitrarily. Consider the current state of a Markov chain is given by the d-dimensional random vector $\boldsymbol{x}_i = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^T$. Here, we assign an arbitrary proposal density for each component as $q_j(x_j|x_j^{(i)})$ and proceed as follows.

1. Generate the first component of the new state, x_1^* using the proposal density $q_1(x_1)$
2. Accept or reject this component using the Metropolis-Hastings acceptance rejection criterion with

$$r(x_1^*, x_2^{(i)}, \ldots, x_d^{(i)}) = \frac{p(x_1^*, x_2^{(i)}, \ldots, x_d^{(i)})}{p(\boldsymbol{x}^{(i)})} \frac{q_1(x_1^{(i)}|x_1^*)}{q_1(x_1^*|x_1^{(i)})} \tag{41}$$

   If accepted, set $x_1^{(i+1)} = x_1^*$, otherwise set $x_1^{(i+1)} = x_1^{(i)}$
3. Generate the second component of the new state, x_2^* using the proposal density $q_2(x_2)$
4. Accept or reject this component using the Metropolis-Hastings acceptance rejection criterion with

$$r(x_2^*, x_1^{(i+1)}, x_3^{(i)} \ldots, x_d^{(i)}) = \frac{p(x_1^{(i+1)}, x_2^*, x_3^{(i)} \ldots, x_d^{(i)})}{p(x_1^{(i+1)}, x_2^{(i)}, x_3^{(i)} \ldots, x_d^{(i)})} \frac{q_2(x_2^{(i)}|x_2^*)}{q_2(x_2^*|x_2^{(i)})} \tag{42}$$

   If accepted, set $x_2^{(i+1)} = x_2^*$, otherwise set $x_2^{(i+1)} = x_2^{(i)}$
5. Generate the j^{th} component of the new state, x_j^* using the proposal density $q_j(x_j)$
6. Accept or reject this component using the Metropolis-Hastings acceptance rejection criterion with

$$r(x_j^*, x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)} \ldots, x_d^{(i)}) = \frac{p(x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_j^*, x_{j+1}^{(i)} \ldots, x_d^{(i)})}{p(x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_j^{(i)}, x_{j+1}^{(i)} \ldots, x_d^{(i)})} \frac{q_j(x_j^{(i)}|x_j^*)}{q_j(x_j^*|x_j^{(i)})} \tag{43}$$

   If accepted, set $x_j^{(i+1)} = x_j^*$, otherwise set $x_j^{(i+1)} = x_j^{(i)}$
7. Continue for all components up to component x_d.

This algorithm applies for any general joint probability distribution $p(\boldsymbol{x})$. When the variables of the random vector $\boldsymbol{X}$ are independent, and therefore $p(\boldsymbol{x}) = \prod_{j=1}^d p_j(x_j)$, the algorithm simplifies considerably and requires only marginal probability density evaluations such that each component is independently updated as:

1. Generate the j^{th} component of the new state, x_j^* using the proposal density $q_j(x_j)$
2. Accept or reject this component using the Metropolis-Hastings acceptance rejection criterion with

$$r(x_j^*, x_j^{(i)}) = \frac{p_j(x_j^*)}{p_j(x_j^{(i)})} \frac{q_j(x_j^{(i)}|x_j^*)}{q_j(x_j^*|x_j^{(i)})} \tag{44}$$

   If accepted, set $x_j^{(i+1)} = x_j^*$, otherwise set $x_j^{(i+1)} = x_j^{(i)}$

7.1.4 Intuition, Diagnostics, Convergence, and Scaling for Random Walk MCMC

Here, we want to first develop an intuition for how random walk MCMC works. We then discuss some of the issues that can arise when applying random walk MCMC methods and how we can combat these issue. We then discuss statistical and other diagnostics that allow us to assess the quality of the generated MCMC samples. Finally, we discuss how to carefully select the proposal density to obtain the best quality MCMC samples for a given problem. In particular, we investigate optimal scaling for the proposal density to obtain the right balance between acceptance and rejection.

Intuitive explanation of random walk Metropolis-Hastings

Consider that we aim to generate samples from an arbitrary distribution as illustrated in Figure **??**. At any stage of the MCMC algorithm, consider that the Markov chain lies in state $x^{(k)}$. We define the symmetric proposal density $q(x|x^{(k)}$ as illustrated by the red curve in Figure **??** and draw a new proposed state x^* as shown. The Metropolis-Hasting acceptance density tells us that we always accept the newly proposed state if the state has higher probability density than the previous state $p(x^*) > p(x^{(k)})$. This can be viewed as an "uphill" move. In this way, we can say that we will always accept samples that take us uphill to a more probable state.

On the other hand, if the newly proposed state has lower probability density than the previous state $p(x^*) < p(x^{(k)})$, then we accept the new state with probability $p(x^*)/p(x^{(k)})$. This can be viewed as a "downhill" move. In other words, if the newly proposed state moves us downhill, to region of lower probability density then the probability of acceptance will depend on how far downhill the proposed state lies. If the new state takes us to a point with much lower probability than the current state, then it will have a high probability of rejection. However, if the newly proposed state has only marginally lower probability density than the current state it will have a high probability of being accepted.

Common issues in random walk MCMC

There are two main issues that arise in random walk MCMC methods. The first issue arises at the initialization of MCMC sampling and depends entirely on the initial state, $x^{(0)}$. If the initial state lies at a point of very low probability density, then the random walk will take many steps to move into regions of high probability density. These initial MCMC steps, which are referred to as *burn-in* will not be representative of the stationary distribution because they lie in regions of extremely low probability. As a result, it is common to discard these initial samples and only retain states once they become representative of the stationary distribution.

The second issue that arises is that the states of the Markov chain are dependent. In fact, it is quite common for nearby states in MCMC to be strongly correlated to one another. This creates two issues: 1. The quality of Monte Carlo estimators improves as the samples become closer to independent; 2. Strongly correlated MCMC chains require a larger number of states to adequately represent the stationary distribution. One way to reduce correlation among the MCMC samples is to apply a *jumping width* in which states are only retained at a fixed interval. That is, only one state out of every n_j states is retained and all other states are discarded where n_j is referred to as the jumping width. This jumping reduces correlation of the states in the Markov chain but greatly increases the number of states that must be simulated; in order to retain N samples, we must simulate $n_j \times N$ states.

MCMC diagnostics

It's not often easy to determine whether we have a quality sample set after completion of the MCMC sampling. To assess the quality of teh sample set, we need a toolbox of diagnostics that allow us to evaluate the quality in different ways. In this section, we will look at a number of both qualitative and quantitative diagnostics that we can use to assess MCMC samples. However, regardless of the diagnostic

used to assess convergence, correlation, and other properties of the chain we recognize that assessment of the MCMC quality often requires a very large number of samples and therefore it is usually beneficial to run the MCMC sampler for as long as possible.

The simplest, and often times the first means of evaluating MCMC samples is to simply plot traces of the Markov chain states. These trace plots will often show basic features of the chains that we can observe visually such as burn-in (Figure **??**a), strong correlations in the chain (Figure **??**b), high rejection rate (Figure **??**c). These trace plots can be very useful to in determining when burn-in is complete or simply visualizing whether the proposal density results in acceptance rates that are either too high or too low. From these trace plots, it may then be helpful to visualize (plot) the autocorrelation function for the chain. If the correlation is strong over short lags and also decays slowly over large lags, then the MCMC method is not mixing well and likely the scale of the proposal density is too small. Examples of the correlation plots for each of the trace plots discussed above are shown in Figure **??**d-f.

Although trace and correlation plots can be informative, it is often helpful to have quantitative statistical measures of sample quality. One such measure is the *effective sample size* (ESS). The ESS gives an estimate of the number of independent samples, N_{ESS}, that has the same estimation power as the set of N correlated samples from MCMC. That is, Monte Carlo estimates from MCMC samples will have variance that is proportional to $1/N_{\text{ESS}}$ rather than $1/N$.

The ESS depends on the autocorrelation function of the Markov chain, which can be computed as

$$\rho(\tau) = \frac{1}{\sigma^2} \int_X (x^{(t)} - \mu)(x^{(t+\tau)} - \mu) p(x) dx \tag{45}$$

where τ denotes the time lag and integration over X implies an integral over all possible states of the Markov chain. This can be reduced to

$$\rho(\tau) = \frac{1}{\sigma^2} \int_X x^{(t)} x^{(t+\tau)} p(x) dx \tag{46}$$

because $x^{(t)}$ and $x^{(t+\tau)}$ have the same marginal distribution as states of the same Markov chain. The ESS is then defined by

$$N_{\text{ESS}} = \frac{N}{\sum_{\tau=-\infty}^{\infty} \rho(\tau)} = \frac{N}{1 + 2\sum_{\tau=0}^{\infty} \rho(\tau)} \tag{47}$$

Estimating ESS therefore requires estimation of the autocorrelation function. In simple cases with long chain lengths, this may be done using standard statistical estimators. In other cases, such as when multiple chains evolve simultaneously, more advanced statistical estimators are required. Some discussion of this will follow in Section 7.2.3.

Finally, these measures of convergence beg the question, "When is the sample size large enough that we can stop sampling?" This is typically done by setting the number of steps of the Markov chain a priori and assessing the ESS, correlations, etc. after completion. However, it is possible to integrate these measures into the procedure and terminate the simulation when, for example, N_{ESS} reaches a specified number.

Optimal scaling

In random walk Metropolis-Hastings MCMC methods, selection of the proposal density $q(\boldsymbol{x}^*|x^{(i)})$ is critical. Poor selection of the proposal density can lead to poor performance of the sampler. Of course, the optimal proposal density would naturally be the joint distribution $p(\boldsymbol{x})$ itself, but we assume that this distribution cannot be directly sampled. Instead, samples are typically drawn from a *symmetric* proposal density having mean value $\boldsymbol{x}^{(i)}$ such that the proposed state can be expressed as $\boldsymbol{x}^* = \boldsymbol{x}^{(i)} + \boldsymbol{z}^{(i)}$ where $\boldsymbol{z}^*$ are *iid* increments drawn from a fixed symmetric zero-mean probability distribution (e.g. $\boldsymbol{z}^* \sim N(0, \sigma^2\mathbf{I})$). The question then become how to select the appropriate scale for this density, i.e. select σ. If σ is too

small, the acceptance rate will be high, but the samples will have strong correlations and will not "mix" well. If σ is too large, the chain will have low acceptance rate and it will rarely move. We therefore seek a scale that balances between these two extremes.

Perhaps the simplest way to look at the scaling is through the acceptance rate. If the acceptance rate is too high, i.e. near 1, then σ is likely too small and the chain will take very small steps. If the acceptance rate is too small, i.e. near 0, then σ is likely too large and the chain will seldom move from its current state. So, a simple rule of thumb is to avoid scalings that yield acceptance rate close to the extremes of 0 and 1. But, this leaves a lot of ambiguity and provides only weak guidance. In fact, under certain simplifying assumptions we can make precise statements about the optimal acceptance rate.

Stationary distribution with iid components

If we consider that the stationary distribuiton of the Markov chain can be expressed as a product density having identical marginals, i.e.

$$p(\boldsymbol{x}) = \prod_{j=1}^{d} f(x_j). \tag{48}$$

That is, the stationary distribution is composed of *iid* components. Of course, this is a trivial case because this simply requires drawing one-dimensional samples from $f(x)$, which is usually easy to do. Nonetheless, Roberts et al. [2] showed that in this simple case, as $d \to \infty$ the *optimal acceptance rate is precisely 0.234*. Numerical studies have suggested that this approximation works reasonably well for d as small as 5, but show that the optimal acceptance rate increases as d decreases such tha the optimal acceptance rate for $d = 1$ is approximately 0.44 [3]. Additional details can be found in the *Handbook of Markov Chain Monte Carlo* [4].

Non-homogeneous normal stationary distribution

Consider the case where the stationary distribution is normally distributed with zero mean and covariance Σ, i.e. $\boldsymbol{x} \sim N(0, \Sigma)$, and the increment random variable $\boldsymbol{z}^*$ is also normally distributed with zero mean and covariance Σ_p, i.e. $z \sim N(0, \Sigma_p)$. In this case, it is best to assume that Σ_p is proportional to Σ, i.e. $\Sigma_p = k\Sigma$ where $k > 0$ is a proportionality constant. Under these conditions, it is optimal to set

$$\Sigma_p = \left[\frac{(2.38)^2}{d}\right] \Sigma \tag{49}$$

This is perhaps a somewhat more useful result because we can perhaps approximate the sampling density as $\sim N(0, \Sigma)$ by simply removing the mean and estimating Σ. This motivates the study of adaptive methods next which will aim to identify the optimal scaling as the chain evolves. Again, additional details on the optimal scaling results presented above can be found in the *Handbook of Markov Chain Monte Carlo* [4].

7.1.5 Adaptive Metropolis and Delayed Rejection

Even if we know the optimal acceptance rate, it's not necessarily straightforward to identify the proposal density that yields this optimal acceptance rate. To identify the optimal proposal density, a *trial and error* process is commonly used. This may be as simple as setting σ, observing the acceptance rate after some period of time and adjusting accordingly. Or it may be as complex as selecting Σ_p, observing the covariance of the resulting sample set after some time, and adjusting Σ_p in accordance with Eq. (49). However, this can be costly and inefficient, especially in cases such as Bayesian inference and reliability where a computationally expensive model needs to be evaluated for every step of the Markov chain or when d becomes large.

An alternative approach is to adaptively refine the proposal density such that $q^{(i)}(\cdot)$ is the proposal density at step i and is selected according to some updating criterion that could depend on the entire prior history of the chain, only the current state of the chain, or anything inbetween. Careful design of the updating scheme and the allowable proposal densities can increase the convergence rate. However, this adaptation could also destroy the convergence by violating important properties, in particular ergodicity, of the Markov chain. That is, even if each proposal density has the correct stationary distribution adaptive algorithms based on these proposals do not necessarily possess the correct stationary distribution.

It is possible, however to establish basic conditions under which convergence can be ensured. Roberts and Rosenthal [5] showed that two conditions must be satisfied. Qualitatively, the first condition – referred to as *diminishing adaptation* – says that the degree of adaptation must go to zero at step i as $i \to \infty$. The second condition – referred to as *containment* – states that the convergence time for all proposal densities must be bounded in probability. More details can be found in the *Handbook of Markov Chain Monte Carlo* [4] where it is argued that the containment condition can be satisfied fairly easily, but care must be taken to satisfty the diminishing adaptation condition.

Adaptive Metropolis

Exploiting the optimality condition in Eq. (49), Haario et al. [6] proposed an adpative Metropolis algorithm that is both simple to use and has been shown to satisfy both the containment and diminishing adaptation conditions under fairly weak conditions. The concept is simple, in each iteration the proposal density is given by

$$\boldsymbol{x}^* \sim N\left(\boldsymbol{x}_i, \left[\frac{(2.38)^2}{d}\right]\Sigma_i + \epsilon \mathbf{I}_d\right) \tag{50}$$

where Σ_i is the estimated covariance matrix for the Markov chain from all states up to $\boldsymbol{x}_i$, i.e. $\{\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(i)}$, $\epsilon > 0$ is a small value ($\epsilon \ll 1$) and $\mathbf{I}$ is the identity matrix. The second term, $\epsilon \mathbf{I}_d$, is added to avoid any variance component from collapsing to zero. Roberts and Rosenthal [7] proposed a similar adaptive density given by

$$\boldsymbol{x}^* \sim (1-\beta)N\left(\boldsymbol{x}_i, \left[\frac{(2.38)^2}{d}\right]\Sigma_i\right) + \beta N(\boldsymbol{x}_i, \Sigma_0) \tag{51}$$

where $0 < \beta < 1$ is a constant and Σ_0 is a positive semi-definite covariance matrix such as $\Sigma_0 = [(0.1)^2/d]\mathbf{I}_d$. In either case, the algorithm starts with a fixed proposal density for a sufficient number of iterations until the covariance matrix Σ_i can be estimated from the generated states.

Delayed Rejection

Another approach to improve convergence is referred to as *delayed rejection*. The simple idea is, upon rejection of a state, rather than set the new state to the prior state another new state is proposed. This allows us to make several tries at each step of the Markov chain and thereby improve convergence. Of course, for each proposed candidate state the reversibility of the Markov chain must be ensured. This can be done by explicitly designing the acceptance density at each stage to satisfy reversibility as was done by Tierney and Mira [8] and outlined by Mira [9].

In this scheme, the first stage is to apply the standard Metropolis Hastings with proposal density $q_1(\boldsymbol{x}_1)$ having acceptance density given in Eq. (28) (and referred to here in as $\alpha_1(\cdot)$) to generate a candidate state $\boldsymbol{x}_1^*$. Rejection of $\boldsymbol{x}_1^*$ suggests that the proposal density should be updated. This update can be made to include the information of the rejected sample, resulting in a new proposal density, $q_2(\boldsymbol{x}_2, \boldsymbol{x}_1^*)$. This new proposal density is used to generate a new state $\boldsymbol{x}_2^*$. To satisfy detailed balance, Tierney and Mira [8] suggested the following acceptance density

$$\alpha_2(\boldsymbol{x}_i, \boldsymbol{x}_1^*, \boldsymbol{x}_1^*) = \min\left(1, \frac{p(\boldsymbol{x}_2^*)q_1(\boldsymbol{x}_2^*, \boldsymbol{x}_1^*)q_2(\boldsymbol{x}_2^*, \boldsymbol{x}_1^*, \boldsymbol{x}_i)[1-\alpha_1(\boldsymbol{x}_2^*, \boldsymbol{x}_1^*)]}{p(\boldsymbol{x}_i)q_1(\boldsymbol{x}_i, \boldsymbol{x}_1^*)q_2(\boldsymbol{x}_i, \boldsymbol{x}_1^*, \boldsymbol{x}_2^*)[1-\alpha_1(\boldsymbol{x}_i, \boldsymbol{x}_1^*)]}\right) \tag{52}$$

In each stage of the delayed rejection scheme, we can express the acceptance density as

$$\alpha_i(\boldsymbol{x}_i, \boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_{i-1}^*) = \min\left(1, \frac{N_i}{D_i}\right). \tag{53}$$

If the i^{th} stage has been reached, it means $i-1$ candidate states $\boldsymbol{x}_j^*$ have been rejected and that $N_j < D_j$ for all $j = 1, 2, \ldots, i-1$. In general, N_i can be determined as

$$N_i = p(\boldsymbol{x}_i^*)q_1(\boldsymbol{x}_i^*, \boldsymbol{x}_{i-1}^*)q_2(\boldsymbol{x}_i^*, \boldsymbol{x}_{i-1}^*, \boldsymbol{x}_{i-2}^*)\cdots q_i(\boldsymbol{x}_i^*, \boldsymbol{x}_{i-1}^*, \ldots, \boldsymbol{x}_i)\cdots \\ [1-\alpha_1(\boldsymbol{x}_i^*, \boldsymbol{x}_{i-1}^*)][1-\alpha_2(\boldsymbol{x}_i^*, \boldsymbol{x}_{i-1}^*, \boldsymbol{x}_{i-2}^*)]\cdots[1-\alpha_{i-1}(\boldsymbol{x}_i^*, \ldots, \boldsymbol{x}_1^*)] \tag{54}$$

and D_i can then be obtained from the following recursive relations

$$D_i = q_i(\boldsymbol{x}_i, \boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_{i-1}^*)[D_{i-1} - N_{i-1}] \tag{55}$$

These results were obtained by directly imposing detailed balance at each stage, essentially enforcing that the rejected states are also rejected in the reverse path. However, this is not strictly necessary. Green and Mira provide an alternate scheme that takes advantage of the rejected variables being integrated out when satisfying detailed balance [10]. Furthermore, as shown by Mira [9], the delayed rejection scheme simplifies considerably when using a symmetric proposal distribution that depends only on the most recently rejected state.

Delayed Rejection Adaptive Metropolis (DRAM)

Haario et al. [11] proposed to combine the delayed rejection (DR) scheme with adaptive Matropolis (AM) in a a method referred to as DRAM. The DRAM method combines DR and AM in a relatively straightforward way. At each new state, i.e. at stage 1 of the DR, the proposal density is selected according to the standard AM. That is, the proposal density is determined by estimating the covariance matrix from all prior accepted states and using Eq. (50). Then, each time a state is rejected the covariance of the proposal density is scaled. That is, at the j^{th} stage of the DR scheme at state i of the Markov chain, the covariance is determined by

$$\Sigma_j^{(i)} = \gamma_j \Sigma_1^{(i)} \tag{56}$$

where the scale factors $\gamma_j > 1$ can be selected arbitrarily. That is, the covariance can be larger or smaller than the AM at prior stages. Haario et al. [11] report that it is generally better to reduced the variance (i.e. decrease γ_j) at later stages.

7.2 Multi-chain Random Walk Methods

Several methods have been developed to concurrently run multiple Markov chains that collectively possess the desired stationary distribution. The advantage of these approach is that they can greatly enhance exploration of the distribution – allowing for example sampling from multi-modal, heavy-tailed, or otherwise complex distributions where single chains may either get stuck sampling from a single mode or require such large proposal densities that the acceptance rate is unacceptably low. However, care must be taken to ensure that the algorithm satisfies detailed balance as some popular methods, such as the Shuffled Complex Evolution Metropolis algorithm [12], violate these properties.

7.2.1 Differential Evolution Adaptive Metropolis (DREAM)

Differential Evolution

The differential evolution Markov chain (DE-MC), proposed by ter Braak [13], views MCMC as a sort of genetic algorithm. Consider a set of N different Markov chains running concurrently, in parallel. Consider that the current state of Markov chain j is given by the d-dimensional vector $\boldsymbol{x}_j^{(i)}$ for $j = 1, \ldots, N$. At the current state, the set of Markov chains collectively form a "population" that can be assembled into a $N \times d$ matrix $\boldsymbol{X}^{(i)}$. For the evolution of each chain $j = 1, \ldots, N$, we first select two members of the population at random, $\boldsymbol{x}_{p_1}^{(i)}$ and $\boldsymbol{x}_{p_2}^{(i)}$ by randomly drawing the index p_1 from the index set $\{1, \ldots, j-1, j+1, \ldots, N\}$ without replacement and independently drawing p_2 from the index set $\{1, \ldots, j-1, j+1, \ldots, N\}$ without replacement. We then propose to evolve the chain by

$$\boldsymbol{x}_j^* = \boldsymbol{x}_j^{(i)} + \gamma(\boldsymbol{x}_{p_1}^{(i)} - \boldsymbol{x}_{p_2}^{(i)}) + \boldsymbol{\epsilon}, \qquad p_1 \neq p_2 \neq j. \tag{57}$$

where γ is a scale factor that is recommended as $\gamma = 2.38/\sqrt{2d}$ with probability 0.9 based on the optimal scaling above for random walk Metropolis-Hastings and $\gamma = 1$ with probability 0.1 to allow for mode jumping, and $\boldsymbol{\epsilon} \sim N(0, \boldsymbol{s})$ is a d-dimensional normal random vector having diagonal covariance $\boldsymbol{s}$ with small standard deviations $s_k \sim O(10^{-6}), k = 1, \ldots, d$. The new state $\boldsymbol{x}_j^*$ is then accepted or rejected according to the standard Metropolis-Hastings acceptance probability as

$$\alpha(\boldsymbol{x}_j^*, \boldsymbol{x}_j^{(i)}) = \min\left(1, \frac{p(\boldsymbol{x}_j^*)}{p(\boldsymbol{x}_j^{(i)})}\right) \tag{58}$$

For each proposed state, the samples $\boldsymbol{x}_{p_1}^{(i)}$ and $\boldsymbol{x}_{p_2}^{(i)}$ are referred to as the parents. In this way, the DE-MC works by determining, at each step of the algorithm, whether the new sample $\boldsymbol{x}_j^*$ should replace its parents in the population. Proof that DE-MC possess the correct stationary distribution can be found in [13].

DREAM

The Differential Evolution Adaptive Metropolis (DREAM) [14] adds several improvements to the DE-MC method. First, the algorithm allow for multiple pairs of parents to be selected in a given iteration. Let δ be the number of pairs of parents that are selected such that $\delta = 1$ in DE-MC. Vrugt [15] suggest drawing δ from the set $\{1, 2, 3\}$ with equal probability 1/3. This means that 1/3 of the time, a single pair will be used, 1/3 of the time two pairs will be used, and 1/3 of the time three pairs will be used. Selecting multiple pairs in this way increases diversity in the proposed states.

Next, DREAM uses an adaptive randomized subspace sampling. That is, it does not update all dimensions in every step, but rather updates only a random subset of the dimensions. In higher dimensions, it is often not optimal to update every dimension in every step. DREAM updates each dimension with probability cr by defining a geometric sequence of n_{CR} crossover probabilities given by

$$CR = \{\tfrac{1}{n_{CR}}, \tfrac{2}{n_{CR}}, \ldots, 1\} \tag{59}$$

At a given iteration, the crossover probability, cr, is drawn randomly from this set with equal probability. Then, in each iteration each dimension is updated with probability cr. Therefore, in each iteration only $d' \leq d$ components of the random vector are updated.

Practically, DREAM works as follows. At each iteration, for the k^{th} dimension of chain j a new state is proposed according to:

$$\boldsymbol{x}_{j,k}^* = \begin{cases} \boldsymbol{x}_{j,k}^{(i)} + (1 + \lambda_k)\gamma(\delta, d') \sum_{l=1}^{\delta} (\boldsymbol{x}_{p_1(l),k}^{(i)} - \boldsymbol{x}_{p_2(l),k}^{(i)}) & \text{if } U_j \leq cr \\ \boldsymbol{x}_{j,k}^{(i)} & \text{if } U_j > cr \end{cases} \tag{60}$$

where $U_j \sim U(0,1)$, $\lambda_k \sim U(-c,c)$ with c typically set equal to 0.1, and the indices p_1, p_2 denote the indices of the δ pairs of parents which are randomly selected (with replacement) from the indices $1,\ldots,N$ corresponding to the N members of the current population. Finally, we notice that the factor $\gamma(\delta, d')$ depends on both the number of pairs of parents, δ and the effective dimension of the update d' and takes the following recommended form

$$\gamma(\delta, d') = \frac{2.38}{\sqrt{2\delta d'}} \tag{61}$$

derived from the optimal scaling relations in Eq. (49).

An additional benefit of the subspace sampling is that it allows differential evolution to be used when $N < d$. In standard differential evolution, the search space is limited to dimension $N-1$ because any set of N points (the population) lies in dimension $N-1$. This means, for example, that if we have $N = 2$ points in $d = 3$ dimensional space, then differential evolution is confined to explore along the line that connects the two points and will only deviate from this line by the small amount ϵ in Eq. (57). This is not sufficient to ensure adequate exploration of the space. The subspace sampling strategy continuously introduces new directions that the chain can explore, even when $N < d$.

A number of other steps are taken during the burn-in period to improve convergence – some of which violate detailed balance and are therefore only used during burn-in. However, these details will not be discussed further here.

7.2.2 Affine Invariant Ensemble Samplers

Recall from Module 3.1 that the *Change of Variables Theorem* tells us that the form of the distribution is preserved under affine transformations of the form $Y = aX + b$. That is, the distribution $p(y)$ can be expressed as

$$p(y) = p(ax + b) \propto p(x). \tag{62}$$

In other words, $p(y)$ and $p(x)$ differ only by a scaling. Recall this property is referred to as *affine invariance*.

The challenge in MCMC is that certain affine transformations may result in poor scaling. This make sampling very difficult using traditional methods like Metropolis-Hastings. Consider, for example, the poorly scaled probability density function

$$p(\boldsymbol{x}) \propto \exp\left(\frac{-(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right) \tag{63}$$

where ϵ is a small value illustrated in Figure 6. Connor - Add this figure. This distribution is difficult to sample using Metropolis-Hastings because only small, local moves with scale $\sqrt{\epsilon}$ in either direction are likely to be accepted. However, if we consider the simple affine transformation given by

$$y_1 = \frac{(x_1 - x_2)}{\sqrt{\epsilon}}, \qquad y_2 = x_1 + x_2 \tag{64}$$

then the distribution becomes well scaled as

$$p(\boldsymbol{y}) = \exp\left(\frac{-(y_1 + y_2)^2}{2}\right). \tag{65}$$

Sampling from $p(\boldsymbol{y}$ is comparatively straightforward.

An affine invariant sampler is one that samples equally well for any two distributions, $p(x)$ and $p(y)$, that are affine invariant. Formally stated, consider MCMC samplers where $X_{t+1} = f(X_t, \xi_t, p(\boldsymbol{x}))$ where

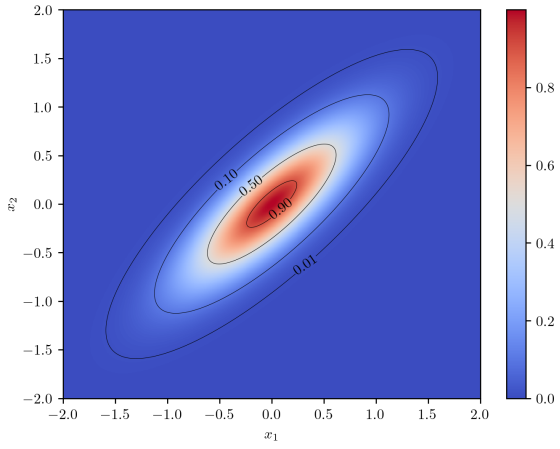


Figure 6: Illustration of the distribution. Uploaded heatmap / contour plot

X_t is the state of the Markov chain at time t, xi_t is a random variable, and $p(\boldsymbol{x})$ is a probability density. This sampler will be *affine invariant* if, for any affine transformation $\boldsymbol{Y} = A\boldsymbol{X} + b$ we have:

$$f(\boldsymbol{y}_t, \xi_t, p(\boldsymbol{y})) = Af(\boldsymbol{x}_t, \xi_t, p(\boldsymbol{x})) + b, \quad \forall \boldsymbol{x} \tag{66}$$

If we consider sampling from these distributions by generating a sequence $\xi_t, t = 0, 1, \ldots$ and starting the chains at $\boldsymbol{x}_0$ and $\boldsymbol{y}_0 = A\boldsymbol{x}_0 + b$, then the two chains will be related by $\boldsymbol{y}_t = A\boldsymbol{x}_t + b$.

There are no known single chain MCMC methods that possess the affine invariance property. Goodman and Weare [16] proposed a family of *ensemble* or multi-chain methods that satisfy the affine invariance property. An *ensemble*, denoted by $\vec{\boldsymbol{X}}$ is composed of a set of L *walkers* $\boldsymbol{X}^{(k)}$ such that $\vec{\boldsymbol{X}} = (\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(L)})$ having target probability density given by:

$$P(\vec{\boldsymbol{x}}) = \prod_{i=1}^{L} p(\boldsymbol{x}^{(i)}) \tag{67}$$

assuming independent walkers, each having distribution $p(\boldsymbol{x})$. Nonetheless, the ensemble Markov chain can satisfy the product density in Eq. (67) without each walker being independent, or even Markovian. This is because the distribution of $\boldsymbol{X}_t^{(k)}$ can depend on $\boldsymbol{X}_{t-1}^{(j)}$. That is, the state of walker k can depend on the previous state of a different walker j.

An affine transformation is applied to the ensemble by applying it separately to each walker. That is:

$$\vec{\boldsymbol{Y}} = A\vec{\boldsymbol{X}} + b = (A\boldsymbol{X}^{(1)} + b, A\boldsymbol{X}^{(2)} + b \ldots, A\boldsymbol{X}^{(L)} + b) \tag{68}$$

The ensemble MCMC method is then affine invariant if the sequence $\vec{\boldsymbol{X}}_t, t = 0, 1, \ldots, T$ drawn from $p(\boldsymbol{x})$ and the sequence $\vec{\boldsymbol{Y}}_t, t = 0, 1, \ldots, T$ satisfy Eq. (68).

To develop a practical affine invariant ensemble sampler, let each walker $\boldsymbol{X}^{(k)}$ be updated using the current positions of all the other walkers. The other walkers form the *complementary ensemble* denoted by

$$\vec{\boldsymbol{X}}_t^{(\sim k)} = (\boldsymbol{X}_{t+1}^{(1)}, \ldots, \boldsymbol{X}_{t+1}^{(k-1)}, \boldsymbol{X}_t^{(k+1)}, \ldots, \boldsymbol{X}_t^{(L)}) \tag{69}$$

This means that the transition probability for walker $\boldsymbol{X}^{(k)}$ will be dependent on both the current state of walker k, $\boldsymbol{x}_t^{(k)}$ and the state of the complementary ensemble, $\vec{\boldsymbol{x}}_t^{(\sim k)}$, expressed as $q(\boldsymbol{x}_{t+1}^{(k)} | \boldsymbol{x}_t^{(k)}, \vec{\boldsymbol{x}}_t^{(\sim k)})$.

The proposed single walker moves are based on partial resampling, which states that the transformation $\vec{\boldsymbol{X}}_t \to \vec{\boldsymbol{X}}_{t+1}$ preserves the joint distribution $P(\vec{\boldsymbol{x}})$ if the single walker moves $\boldsymbol{X}_t^{(k)} \to \boldsymbol{X}_t^{(k+1)}$ preserves the conditional distribution $p(\boldsymbol{x}^{(k)}|\vec{\boldsymbol{x}}_t^{(\sim k)})$. Given that we consider the joint density $P(\vec{\boldsymbol{x}})$ to have the form in Eq. (67) with independent walkers, this means that $q(\boldsymbol{x}_{t+1}^{(k)}|\boldsymbol{x}_t^{(k)}, \vec{\boldsymbol{x}}_t^{(\sim k)})$ preserves $p(\boldsymbol{x}^{(k)})$ for all $\vec{\boldsymbol{x}}^{(\sim k)})$. This is achieved by satisfying detailed balance through a proposal density and a Metropolis-Hastings like acceptance rejection.

Although Goodman and Weare [16] suggest three types of moves, the *stretch*, *walk*, and *replacement* moves, we will focus here only the the *stretch* move and refer to [16] for the others. To move walker $\boldsymbol{X}^{(k)}$, the stretch move randomly selects one additional walker from the complementary ensemble, $\boldsymbol{X}^{(j)} \in \vec{\boldsymbol{X}}_t^{(\sim k)}, j/nek$ and proposes a move of the following form

$$\boldsymbol{X}_*^{(k)} = \boldsymbol{X}^{(j)} + Z(\boldsymbol{X}_t^{(k)} - \boldsymbol{X}^{(j)}) \tag{70}$$

where Z is a random variable having probability density satisfying the symmetry condition $g(1/z) = zg(z)$ to make the move symmetric having $p(\boldsymbol{X}_t^{(k)} \to \boldsymbol{X}_*^{(k)}) = p(\boldsymbol{X}_*^{(k)} \to \boldsymbol{X}_t^{(k)})$. Goodman and Weare [16] suggest a density of the following form

$$g(z) \propto \begin{cases} \dfrac{1}{\sqrt{z}} & \text{if } z \in \left[\dfrac{1}{a}, a\right] \\ 0 & \text{otherwise} \end{cases} \tag{71}$$

where the parameter $a > 1$ is selected to optimize performance. They then show, using partial resampling, that the candidate sample should be accepted, $\boldsymbol{X}_{t+1}^{(k)} = \boldsymbol{X}_*^{(k)}$ with probability

$$\alpha = \min\left(1, \frac{\|\boldsymbol{X}_*^{(k)} - \boldsymbol{X}^{(j)}\|^{d-1} p(\boldsymbol{X}_*^{(k)})}{\|\boldsymbol{X}_t^{(k)} - \boldsymbol{X}^{(j)}\|^{d-1} p(\boldsymbol{X}_t^{(k)})}\right) = \min\left(1, Z^{d-1}\frac{p(\boldsymbol{X}_*^{(k)})}{p(\boldsymbol{X}_t^{(k)})}\right). \tag{72}$$

Otherwise, $\boldsymbol{X}_{t+1}^{(k)} = \boldsymbol{X}_t^{(k)}$. The resulting Markov chain satisfies detailed balance.

7.2.3 Diagnostics for multi-chain methods

It can be particularly difficult to assess the quality of MCMC samples when multiple chains are propagated simultaneously. It's is not as straightforward to simply view trace plots of a single chain because the multiple chains often times have dependencies that cannot be seen by viewing each chain individually. For this reason, we must develop statistical metrics that assess sample quality.

As in the single chain case, we can compute *effective sample size* although we must now include the influence of correlations between chains. Consider that we have M chains, each of length N. We can assess the autocorrelation function $\rho(\tau)$ by computing separately the autocorrelation function for each chain, $\hat{\rho}_m(\tau), m = 1, \ldots, M$, the within-chain variance given by

$$W = \frac{1}{M}\sum_{m=1}^{M} s_m^2 \tag{73}$$

where

$$s_m^2 = \frac{1}{N-1}\sum_{t=1}^{N}(x_t^{(m)} - \bar{x})^2 \tag{74}$$

is the variance of chain m and

$$\bar{x} = \frac{1}{M}\sum_{m=1}^{M}\bar{x}_m = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{N}\sum_{t=1}^{N} x_t^{(m)} \tag{75}$$

and the multi-chain (or between chain) variance given by

$$B = \frac{N}{M-1}\sum_{m=1}^{M}(\bar{x}_m - \bar{x})^2. \tag{76}$$

The total variance is then a mixture of the within-chain variance W and the between-chain variance given by

$$\mathrm{Var}(x) = \frac{N-1}{N}W + \frac{1}{N}B \tag{77}$$

The combined autocorrelation is then given by

$$\hat{\rho}(\tau) = 1 - \frac{W - \frac{1}{M}\sum_{m=1}^{M}\hat{\rho}_m(\tau)}{\mathrm{Var}(x)} \tag{78}$$

This combined autocorrelation can be used to replace the estimated single-chain correlation function in the ESS given by Eq. (47). Furthermore, because noise in the autocorrelations increases as t increases, it is common to truncate the autocorrelation at a fixed maximum lag. Moreover, because the odd autocorrelation may be negative, some additional statistical manipulations may be necessary. These are discussed further in several texts including Geyer [17] and the Handbook of Markov Chain Monte Carlo [18].

To a related end, Gelman and Rubin [19] proposed the *potential scale reduction statistic*, $\hat{R}$, that aims to compare multiple chains to determine whether the chains have converged to the stationary distribution. $\hat{R}$ measures the ratio of the average variance within the chains to the variance across chains and is given by

$$\hat{R} = \sqrt{\frac{\mathrm{Var}(x)}{W}} = \sqrt{\frac{\frac{N-1}{N}W + \frac{1}{N}B}{W}} \tag{79}$$

If all chains are at sufficiently sampling across the stationary distribution, then $\hat{R} = 1$. Gelman and Rubin recommend to begin sampling from the M independent chains with initial states that are sufficiently spread across the domain and continue propagating the chains until $\hat{R} < 1.1$.

The Gelman and Rubin statistic is intuitively simple and can be a very useful. However, Flegal et al. [20] note that it can be unstable in certain cases, which can cause the statistic to recommend either very short chains or very long chains.

7.3 Gradient-based Methods

The sample quality of MCMC methods can be greatly enhanced by incorporating gradient information into the proposal density. There are several methods to do so, including the Metropolis Adjusted Langevin Algorithm (MALA) and the important Hamiltonian Monte Carlo that we will discuss here. Although these methods often produce samples of superior quality to standard random walk methods, they come at significant additional expense and some potential challenges because they require numerical computation of the gradients of the stationary density. These gradients can pose both numerical and computational challenges, particularly in Bayesian inference, if existing data are noisy or if evaluation of the log-likelihood involves the evaluation of an expensive numerical model. These computational issues will be revisited in our discussion of Bayesian methods in Module 5.

7.3.1 Metropolis Adjusted Langevin Algorithm (MALA)

The *Metropolis Adjusted Langevin Algorithm* (MALA) proposes new steps through overdamped Langevin dynamics, which models the dynamics of a system of interacting particles as an Itô diffusion process. Accordingly, the state of the system of particles, X_t evolves according to the following stochastic differential equation:

$$dX_t = b(X_t)dt + \sigma(X_t)dB_t \tag{80}$$

where b is a vector field known as the *drift coefficient*, σ is a matrix field known as the *diffusion coefficient*, and B is a Brownian motion. In MALA, this diffusion process is modeled as

$$\dot{X} = \nabla \log(p(\boldsymbol{x})) + \sqrt{2}\dot{B} \tag{81}$$

Since the Itô diffusion process is Markovian, it is natural to model the state transitions in MCMC in this way by simply setting the drift equal to the gradient of the stationary distribution. Under these conditions, it can be shown that the stationary distribution of the diffusion process is, in fact, $p(\boldsymbol{x})$.

MCMC chains can be generated from this diffusion process through time discretization methods such as the Euler-Maruyama method [] having fixed time step Δt. In this way, we set the initial state of the diffusion process (Markov chain) as $\boldsymbol{X}_0 = \boldsymbol{x}_0$ and solve for each subsequent state (discrete time step) as

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \Delta t \nabla \log(p(\boldsymbol{X}_t)) + \sqrt{2\Delta t}\xi_t \tag{82}$$

where $\boldsymbol{\xi_t}$ are independent samples of a d-dimensional standard normal random variable. The new state $\boldsymbol{X}_{t+t}$ is normally distributed with mean $\boldsymbol{X}_t + \Delta t \nabla \log(p(\boldsymbol{X}_t))$ and covariance equal to $\sqrt{2\Delta t}\boldsymbol{I}$ where $\boldsymbol{I}$ is the $d \times d$ identity matrix. In this approach, referred to as the *Unadjusted Langevin Algorithm* (ULA), every newly generated time step represents a new state of the Markov chain. That is, every time step is **accepted**. Besag [21] and later Roberts and Tweedie [22] showed that this can actually have quite poor performance and potentially strong dependents on the time step Δt. They then proposed to add a Metropolis-Hastings acceptance step such that $\tilde{\boldsymbol{X}}_{t+1}$ is generated and represents a *proposed* step, which is accepted with probability

$$\alpha = \min\left\{1, \frac{p(\tilde{\boldsymbol{X}}_{t+1})q(\boldsymbol{X}_t \mid \tilde{\boldsymbol{X}}_{t+1})}{p(\boldsymbol{X}_t)q(\tilde{\boldsymbol{X}}_{t+1} \mid \boldsymbol{X}_t)}\right\}, \tag{83}$$

where,

$$q(x' \mid x) \propto \exp\left(-\frac{1}{4\Delta t}\|x' - x - \Delta t \nabla \log p(x)\|_2^2\right) \tag{84}$$

is the transition probability from x to x', which is not symmetric in general.

The combined Langevin dynamics with Metropolis-Hastings can be shown to satisfy detailed balance and, in general, results in higher acceptance rate because the Langevin dynamics inform the proposal density. It can be shown that, for certain classes of distributions, the optimal acceptance rate is equal to 0.574. If the acceptance rate differs significantly from this, then Δt should be adjusted [23].

7.3.2 Hamiltonian Monte Carlo (HMC)

Hamiltonian Monte Carlo makes the analogy between sampling and a set of particles moving through a probability distribution, $p(\mathbf{x})$, with no external forces (i.e. only conservative forces). Under these conditions, the moving particles are governed by the following principles of Hamiltonian mechanics, wherein the the state of the particles can be fully described by the position vector $\mathbf{q}$ and the momentum vector $\mathbf{p}$. Because the system is conservative, the energy of the system (Hamiltonian) is a constant composed of the kinetic and

potential energy of the system, K and U, which depend only on the position and momentum respectively as follows:

$$H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p}) \ . \tag{85}$$

The state of the system evolves in time using Hamiltonian equations relating the position $\mathbf{q}$ and the momentum $\mathbf{p}$ as:

$$\begin{aligned} \frac{d\mathbf{q}}{dt} &= \frac{\partial H}{\partial \mathbf{p}} \ , \\ \frac{d\mathbf{p}}{dt} &= -\frac{\partial H}{\partial \mathbf{q}} \ . \end{aligned} \tag{86}$$

Evolving this system of equations in time results in a constant energy (Hamiltonian) trajectory in $(\mathbf{p}, \mathbf{q})$ phase space. Importantly, for HMC these equations describe a system that is reversible and symplectic, conserves energy, and preserves volume. The importance of these properties for HMC have been detailed by Neal [24] and will not be discussed further here.

HMC leverages Hamiltonian dynamics by treating the sample $\mathbf{x}$ drawn from $p(\mathbf{x})$ as a set of particles having positions $\mathbf{q}$ in the Hamiltonian trajectory (i.e. $\mathbf{x} \equiv \mathbf{q}$). We then define the joint probability density between the position $\mathbf{q}$ and momentum $\mathbf{p}$ to take the following form

$$p(\mathbf{p}, \mathbf{q}) \propto e^{-H(\mathbf{p},\mathbf{q})} = e^{-U(\mathbf{q})} e^{-K(\mathbf{p})} \tag{87}$$

We then define the potential energy as

$$U(\mathbf{q}) = -\log \pi(\mathbf{q}) \ , \tag{88}$$

and the kinetic energy as:

$$K(\mathbf{p}) = \frac{1}{2} \mathbf{p} \mathbf{M}^{-1} \mathbf{p} \ , \tag{89}$$

which results in the position following the distribution $p(\mathbf{q})$ and the momentum independently following a Gaussian distribution with covariance $\mathbf{M}$. It is common to select $\mathbf{M} = \alpha \mathbf{I}$ where $\mathbf{I}$ is the identity matrix and α is a scalar value, which corresponds to the momenta being indepedent Gaussian random variables. Selecting $\mathbf{M}$ to be a non-diagonal, positive semi-definite covariance matrix results in the Reimannian Manifold HMC, which has been demonstrated to improve the acceptance rate for non-Gaussian distributions but requires determining the best $\mathbf{M}$ matrix for a given problem [25].

To perform HMC, we then consider the current state of the Markov chain as the position $\mathbf{q}$ and assign a Gaussian random momentum $\mathbf{p}$, which completely defines a state in phase space such that Eq. (86) defines a reversible trajectory of constant energy (Hamiltonian). Hamilton's equations (Eq. (86)) are then solved numerically using a symplectic integrator such as the synchonized leapfrog scheme which first updates the positions as

$$q_i(t + \Delta t) = q_i(t) + \frac{\Delta t}{m_i} \ p_i(t) + \frac{\Delta t^2}{2 m_i} \ \dot{p}_i(t) = q_i(t) + \frac{\Delta t}{m_i} \ p_i(t) - \frac{\Delta t^2}{2 m_i} \ \frac{\partial H}{\partial q_i(t)} \tag{90}$$

where Δt is the integration time step and $\dot{p}_i(t) = -\frac{\partial H}{\partial q_i(t)}$. Then, the momenta are updated by

$$p_i(t + \Delta t) = p_i(t) + \frac{\Delta t}{2} \ \left(\dot{p}_i(t) + \dot{p}_i(t + \Delta t) \right) = p_i(t) - \frac{\Delta t}{2} \ \left(\frac{\partial H}{\partial q_i(t)} + \frac{\partial H}{\partial q_i(t + \Delta t)} \right) \tag{91}$$

where again $\dot{p}_i(t) = -\frac{\partial H}{\partial q_i(t)}$.

A new state $(\mathbf{q}^*, \mathbf{p}^*)$ is determined by integrating forward L time steps for a total time of $t_f = L\Delta t$. Because the integration is performed numerically, there is some error that causes a deviation in the Hamiltonian (i.e. H is not strictly conserved). Therefore, a Metropolis-Hastings type acceptance-rejection criterion is introduced that accepts the new state with probability

$$\alpha = \min\left[1, \exp(H((\mathbf{q}, \mathbf{p}) - H(\mathbf{q}^*, \mathbf{p}^*)\right] \ . \tag{92}$$

The HMC process is highly sensitive to both L and Δt such that large Δt can cause large integration errors and thus high rejection rates. Meanwhile, small L can result in undesirable random-walk type behavior while large L can result in unnecessarily high computation cost. To avoid the need to manually tune the parameters, one may apply methods such as the No-U-Turns Sampler (NUTS) [26], which automatically determines the end time of the Hamiltonian dynamics, $T = L\Delta t$.

Interestingly, the MALA algorithm corresponds exactly to HMC in which the Hamiltonian dynamics are integrated for only a single time step.

8 Variance Reduction Techniques

Monte Carlo simulation is a robust method for propagating uncertainty but, as we have seen, its statistical convergence is very slow. Consider that we wish to estimate a given statistical moment, denoted by $\theta = \mathbb{E}[Y] = \mathbb{E}[h(\mathbf{X})]$ where $Y = h(\cdot)$ is a general function or mathematical operation and $\mathbf{X}$ is a random vector of inputs to that function. Denote this Monte Carlo estimator by $\hat{\theta}_N$. Recall from the *Law of Large Numbers* that the variance of the estimator scales as $\mathrm{Var}(\hat{\theta}) = \dfrac{\sigma^2}{N}$ where $\sigma^2 = \mathrm{Var}(Y)$ and N is the number of samples. Consequently, reducing the variance of the estimator requires large sample sizes. To formalize this, we can assess the quality of an estimate generated from Monte Carlo simulation by deriving approximate confidence intervals for $\hat{\theta}_N$.

From the Central Limit Theorem, we can assume that $\hat{\theta}_N$ follows a normal distribution where the $100(1-\alpha)\%$ confidence intervals are derived as follows. First, let us standardize $\hat{\theta}_N$ as

$$Z = \frac{\hat{\theta}_N - \theta}{\hat{\sigma}_N/\sqrt{N}} \tag{93}$$

where $\theta = \mathbb{E}[h(\mathbf{X})]$ is the moment being estimated and $\hat{\sigma}_N$ is a statistical estimator of $\sqrt{\mathrm{Var}(Y)}$ given by:

$$\sigma_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \hat{\theta}_N)^2 \tag{94}$$

with $Y_i = h(\mathbf{X}_i)$. For the $100(1-\alpha)\%$ confidence intervals, we want to determine z such that $P(-z \leq Z \leq z) = 1 - \alpha$. The value of z can be determined from the normal distribution such that

$$\Phi(z) = P(Z \leq z) = 1 - \frac{\alpha}{2} \tag{95}$$

The resulting value, denoted $z_{1-\alpha/2}$ can be used to determine confidence intervals as follows. We first recognize that

$$1 - \alpha = P(-z_{1-\alpha/2} \leq Z \leq z_{1-\alpha/2}) \tag{96}$$

Plugging in Eq. (93) yields:

$$1 - \alpha = P\left(-z_{1-\alpha/2} \leq \frac{\hat{\theta}_N - \theta}{\hat{\sigma}_N/\sqrt{N}} \leq z_{1-\alpha/2}\right) \tag{97}$$

Finally solving for θ gives:

$$1-\alpha = P\left(\hat{\theta}_N - z_{1-\alpha/2}\frac{\hat{\sigma}_N}{\sqrt{N}} \leq \theta \leq \hat{\theta}_N + z_{1-\alpha/2}\frac{\hat{\sigma}_N}{\sqrt{N}}\right) \tag{98}$$

which defines confidence intervals for θ. That is, given N samples the estimate of θ will fall in the bounds

$$\left[\hat{\theta}_N - z_{1-\alpha/2}\frac{\hat{\sigma}_N}{\sqrt{N}}, \hat{\theta}_N + z_{1-\alpha/2}\frac{\hat{\sigma}_N}{\sqrt{N}}\right] \tag{99}$$

$100(1-\alpha)\%$ of the time.

Clearly by adding samples (increasing N), the confidence intervals for θ will narrow. However, they do so slowly by scaling with $\frac{1}{\sqrt{N}}$. We aim to make these confidence intervals as narrow as possible without increasing N to very large values. To do so, we can alternatively reduce $\hat{\sigma}_N$, the estimated standard deviation of Y. Strategies that do so are referred to as *variance reduction techniques*. Recall the standard Monte Carlo estimators from Eq. (8). Let us rewrite this expression such that we replace the model output $h(X)$ with a generic unbiased estimator $\hat{\theta}$ such that we can estimate θ by

$$\theta = \mathbb{E}[Y] \approx \frac{1}{N}\sum_{i=1}^{N}\hat{\theta}_i \tag{100}$$

In traditional Monte Carlo, the estimator is simply the model/simulation output, i.e.

$$\hat{\theta} = Y \tag{101}$$

But there are many other such estimators and we can seek estimators that have lower variance, that is $\mathrm{Var}(\hat{\theta}) < \mathrm{Var}(Y)$. Next, we will explore certain estimators and their properties.

8.1 Control Variates

Consider that we are interested in computing $\theta = \mathbb{E}[Y]$ where $Y = h(X)$ is the output of a simulation model. Suppose that Z is also an output of the model and that $\mathbb{E}[Z]$ is known. Z is known as a *control variate* for Y. Let's now use $\mathbb{E}[Z]$ to construct a new Monte Carlo estimator. Let us define the *control variates* estimator by

$$\hat{\theta}_c = Y - c(Z - \mathbb{E}[Z]) \tag{102}$$

such that we can estimate θ by:

$$\theta \approx \hat{\theta}_{N,c} = \frac{1}{N}\sum_{i=1}^{N} Y_i - c(Z_i - \mathbb{E}[Z]). \tag{103}$$

Here, c is a constant that governs the influence of the second term. Regardless of the value of c, it is straightforward to show that $\mathbb{E}[\hat{\theta}_c] = \theta$.

To obtain a variance reduction, we therefore need to determine whether $\mathrm{Var}(\hat{\theta}_c) < \mathrm{Var}(Y)$. Let us simply take the variance of $\hat{\theta}_c$ from Eq. (102) as:

$$\mathrm{Var}(\hat{\theta}_c) = \mathrm{Var}(Y) + c^2\mathrm{Var}(Z) + 2c\mathrm{Cov}(Y, Z). \tag{104}$$

We can clearly see that we will obtain a variance reduction if

$$c^2\mathrm{Var}(Z) + 2c\mathrm{Cov}(Y, Z) < 0, \tag{105}$$

which depends both on the linear dependence between Y and Z and the value of c. Since we are free to choose c, let us choose it such that it minimizes $\text{Var}(\hat{\theta}_c)$ from Eq. (104). Simple calculus yields the value c^* that minimizes variance as:

$$c^* = -\frac{\text{Cov}(Y, Z)}{\text{Var}(Z)} \tag{106}$$

Using this optimal value in our estimator, the variance can be determined by plugging c^* into Eq. (104)

$$\text{Var}(\hat{\theta}_{c^*}) = \text{Var}(Y) - \frac{\text{Cov}(Y, Z)}{\text{Var}(Z)} \tag{107}$$

We therefore see that the variance will be reduced if $\text{Cov}(Y, Z) \neq 0$. That is, the optimal control variates estimator will reduce variance if Y and Z have *any* linear dependence. Finally, we apply the new control variates estimator to the Monte Carlo estimate for θ to obtain

$$\theta \approx \hat{\theta}_{N,c^*} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - c^*(Z_i - \mathbb{E}[Z])) \tag{108}$$

The main idea of control variates therefore is to leverage additional information from the simulation model to improve the Monte Carlo estimate. However, this additional information must be linearly related to the output of interest, Y. One of the main challenges in applying control variates is that we generally do not know $\mathbb{E}[Z]$, $\text{Var}(Z)$, and $\text{Cov}(Y, Z)$. Therefore, when applying control variates we must also estimate these quantities using Monte Carlo estimates.

Multiple Control Variates

The above formulation assumes that we have one control variate. But, this can be generalized to allow for any number of control variates. Consider that we have m additional outputs from the model, which we will denote by $Z_i, i = 1, \ldots, m$ with known $\mathbb{E}[Z_i]$. We can then construct the following unbiased estimator

$$\hat{\theta}_c = Y + \sum_{i=1}^{m} c_i(Z_i - \mathbb{E}[Z_i]). \tag{109}$$

This, of course, requires us to now determined a set of coefficients c_i that will result in a variance reduction. To determine these values, we begin by expressing the variance of the estimator in Eq. (109) as

$$\text{Var}(\hat{\theta}_c) = \text{Var}(Y) + 2\sum_{i=1}^{m} c_i\text{Cov}(Y, Z_i) + \sum_{i=1}^{m}\sum_{j=1}^{m} c_i c_j \text{Cov}(Z_i, Z_j) \tag{110}$$

To determine the optimal values of c_i, we can once again take the partial derivatives with respect to each c_i and set them equal to zero. This yields a set of linear equations that can be solved using standard regression methods such as least squares.

Multi-level and Multi-fidelity Monte Carlo

A great deal of research in recent years has focused on the integration of information from multiple models having the same output. These works, which are typically defined by considering multiple models of the same processing having different fidelity are referred to as *Multi-fidelity Monte Carlo* methods. In these methods, we typically have a high-fidelity model whose output, Y, whose moments we are interested in estimating, i.e. $\mathbb{E}[Y]$, and a low-fidelity model (or series of low-fidelity models) whose output Z is the same quantity as Y but is generated from a lower quality model. In these cases, estimators are constructed that use Z as a control variate for Y.

8.2 Antithetic Variates

Consider the case where we have $Y = h(\boldsymbol{U})$ where $\boldsymbol{U} = [U_1, U_2, \ldots, U_m]^T$ are *iid* uniform random variables on $(0, 1)$. We wish to estimate $\theta = \mathbb{E}[h(\boldsymbol{U})]$. Using traditional Monte Carlo simulation with $2N$ samples (the reason for using $2N$ samples will become clear later), we have

$$\theta \approx \hat{\theta}_{2N} = \frac{1}{2N}\sum_{i=1}^{2N} Y_i = \frac{1}{2N}\sum_{i=1}^{2N} h(\boldsymbol{U}_i) \tag{111}$$

Alternatively, we could generate N samples of $\boldsymbol{U}$ and set the other N samples equal to $\mathbf{1} - \boldsymbol{U}$ where $\mathbf{1}$ is the vector of ones, noting that for each component U_j since $U_j \sim U(0, 1)$, $1 - U_j$ is also $U(0, 1)$. We can then define the variable $\tilde{Y} = h(\mathbf{1} - \boldsymbol{U}_i)$ and the new estimator:

$$\hat{\theta}_a = \frac{Y + \tilde{Y}}{2} \tag{112}$$

We can clearly see that $\mathbb{E}[\hat{\theta}_a] = 0$ and therefore $\hat{\theta}_a$ can be used to estimate θ. Moreover, if $\boldsymbol{U}$ are *iid*, then so are $\mathbf{1} - \boldsymbol{U}$ and the Law of Large Numbers is valid. Here $\boldsymbol{U}$ and $\mathbf{1} - \boldsymbol{U}$ are called antithetic variates and we can construct the following Monte Carlo estimator

$$\theta \approx \hat{\theta}_{N,a} = \frac{1}{N}\sum_{i=1}^{N} \hat{\theta}_a^{(i)} = \frac{1}{N}\sum_{i=1}^{N} \frac{Y_i + \tilde{Y}_i}{2} = \frac{1}{N}\sum_{i=1}^{N} \frac{h(\boldsymbol{U}_i) + h(\mathbf{1} - \boldsymbol{U}_i)}{2} \tag{113}$$

We now compare the estimators $\hat{\theta}_{N,a}$ and $\hat{\theta}_{2N}$ to determine which has lower variance. We can easily extend our classical Monte Carlo estimator of variance to a sample size of $2N$ as

$$\mathrm{Var}(\hat{\theta}_{2N}) = \mathrm{Var}\left(\frac{\sum_{i=1}^{2N} Y_i}{2N}\right) = \frac{1}{4N^2}\mathrm{Var}\left(\sum_{i=1}^{2N} Y_i\right) = \frac{2N}{4N^2}\mathrm{Var}\left(Y\right) = \frac{1}{2N}\mathrm{Var}\left(Y\right). \tag{114}$$

Furthermore, we can express the variance of $\hat{\theta}_{N,a}$ as

$$\begin{aligned}
\mathrm{Var}(\hat{\theta}_{N,a}) &= \mathrm{Var}\left(\frac{\sum_{i=1}^{N} \hat{\theta}_a^{(i)}}{N}\right) \\
&= \frac{1}{N}\mathrm{Var}(\hat{\theta}_a) \\
&= \frac{1}{4N}\mathrm{Var}(Y + \tilde{Y}) \\
&= \frac{1}{2N}\mathrm{Var}(Y) + \frac{1}{2N}\mathrm{Cov}(Y, \tilde{Y}) \\
&= \mathrm{Var}(\hat{\theta}_{2N}) + \frac{1}{2N}\mathrm{Cov}(Y, \tilde{Y})
\end{aligned} \tag{115}$$

We therefore see that if $\mathrm{Cov}(Y, \tilde{Y}) < 0$, then $\mathrm{Var}(\hat{\theta}_{N,a}) < \mathrm{Var}(\hat{\theta}_{2N})$.

How then can we ensure that $\mathrm{Cov}(Y, \tilde{Y}) < 0$? In fact, following Eq. (115) shows that antithetic variates may actually lead to a variance *increase*. The following Theorem provides some sufficient conditions for which a variance reduction can be guaranteed.

Theorem 1 *If $h(u_1, \ldots, u_m)$ is a monotonic function of each of its arguments on $[0, 1]^m$, then for a set of IID random variables $\boldsymbol{U} = U_1, \ldots, U_n$*

$$Cov(h(\boldsymbol{U}), h(\mathbf{1} - \boldsymbol{U})) < 0.$$

The proof to this theorem can be found in [27]. This theorem provides a set of *sufficient* conditions to guarantee a variance reduction using antithetic variates. However, these conditions are not *necessary*. Indeed, there are other conditions involving non-monotonic functions when a variance reduction can be achieved. We simply do not have guarantees in such cases.

Non-Uniform Antithetic Variates

For general problem in Monte Carlo simulation, where $Y = h(\boldsymbol{X})$, the inputs $\boldsymbol{X}$ are not necessarily uniform random variables. Nonetheless, we can still use the antithetic variates method if $\boldsymbol{X}$ are independent random variables by taking advantage of the *inverse transform*. That is, using the inverse transform method we can express each component X_i of the vector $\boldsymbol{X}$ in terms of a uniform random variable, $U_i \sim U(0,1)$ as $X_i = F_{X_i}^{-1}(U_i)$. We can then express the function $h(\boldsymbol{X})$ as follows

$$Y = h(F_{X_1}^{-1}(U_1), \ldots, F_{X_m}^{-1}(U_m)) \tag{116}$$

We have now expressed the function in terms of uniform random variables for which we can apply antithetic variates.

Furthermore, we can show that Theorem 1 holds in this more general case. Recall that, by definition, the cumulative distribution function is a non-decreasing function. Therefore, it's inverse is also a non-decreasing function. It follows that, if $h(\boldsymbol{X})$ is monotonic in each of its components, then $h(F_{X_1}^{-1}(U_1), \ldots, F_{X_m}^{-1}(U_m))$ is a monotonic function of all components U_i.

Normal Antithetic Variates

As mentioned above, the inverse transform method is not always easily applied in cases where the quantile function cannot be expressed analytically. In the case of normal random variables, we can apply antithetic variates as follows.

Consider the function $Y = h(\boldsymbol{X})$ where $\boldsymbol{X} = [X_1, \ldots, X_m]^T$ is a normal random vector having components $X_i \sim N(\mu_i, \sigma_i)$. Defining the antithetic variate $\tilde{\boldsymbol{X}} = 2\boldsymbol{\mu} - \boldsymbol{X}$ where $\boldsymbol{\mu}$ is the vector of mean values μ_i, we see that $\tilde{\boldsymbol{X}}$ also has components $X_i \sim N(\mu_i, \sigma_i)$. Clearly $\text{Cov}(\boldsymbol{X}, \tilde{\boldsymbol{X}}) < 0$. Therefore, Theorem 1 again applies and a variance reduction will be guaranteed if $h(\boldsymbol{X})$ is monotonic in all of its components.

8.3 Importance Sampling

Consider the random variable X having probability density function $f_X(x)$ and that we aim to estimate $\theta = \mathbb{E}_f[h(X)]$. Note that we now express the expectation as $\mathbb{E}_f[\cdot]$ where the subscript f denotes that the expectation is being taken with respect to density $f_X(x)$. The expectation is expressed

$$\theta = \mathbb{E}_f[h(X)] = \int_{-\infty}^{\infty} h(x) f_X(x) dx \tag{117}$$

Let $q_X(x)$ be another PDF having the same support as $f_X(x)$ (i.e. $q_X(x) \neq 0$ whenever $f_X(x) \neq 0$). We can alternatively express θ by

$$\begin{aligned} \theta &= \int_{-\infty}^{\infty} h(x) \frac{f_X(x)}{q_X(x)} q_X(x) dx \\ &= \mathbb{E}_q \left[h(x) \frac{f_X(x)}{q_X(x)} \right] \end{aligned} \tag{118}$$

We therefore define our importance sampling estimator as

$$\hat{\theta}_{is} = h(x) \frac{f_X(x)}{q_X(x)} \tag{119}$$

and revise our Monte Carlo estimate of θ as

$$\theta \approx \hat{\theta}_{N,is} = \frac{1}{N}\sum_{i=1}^{N} \hat{\theta}_{is}^{(i)} = \frac{1}{N}\sum_{i=1}^{N} \frac{f_X(x_i)}{q_X(x_i)} h(x_i) \tag{120}$$

To use this estimator, we notice that the expectation has been recast in terms of the density $q_X(x)$. We therefore draw random samples from $q_X(x)$ (instead of $f_X(x)$), which we refer to as the *importance sampling density*.

The challenge in applying importance sampling is to identify the best importance sampling distribution that ensures we achieve a variance reduction. To gain an appreciation for how we can reduce variance using importance sampling, let's start by comparing the two estimators $\mathbb{E}_f[h(X)]$ and $\mathbb{E}_q[\hat{\theta}_{is}]$ where we recognize that:

$$\theta = \mathbb{E}_f[h(X)] = \mathbb{E}_q[\hat{\theta}_{is}] \tag{121}$$

We can express the variance of the original function as

$$\text{Var}_f(h(X)) = \int_{-\infty}^{\infty} h(x)^2 f_X(x) dx - \theta^2 \tag{122}$$

where again the subscript f denotes that our expection is taken with respect to the original density $f_X(x)$. Likewise, we can express the variance of the importance sampling density estimator $\hat{\theta}_{is}$ as

$$\begin{aligned}
\text{Var}_q(\hat{\theta}_{is}) &= \int_{-\infty}^{\infty} \hat{\theta}_{is}(x)^2 q_X(x) dx - \theta^2 \\
&= \int_{-\infty}^{\infty} \left(h(x)\frac{f_X(x)}{q_X(x)} \right)^2 q_X(x) dx - \theta^2 \\
&= \int_{-\infty}^{\infty} \frac{h^2(x) f_X(x)}{q_X(x)} f_X(x) dx - \theta^2
\end{aligned} \tag{123}$$

Taking the difference between the estimators in Eqs. (122) and (123) yields

$$\text{Var}_f(h(X)) - \text{Var}_q(\hat{\theta}_{is}) = \int_{-\infty}^{\infty} h^2(x) \left(1 - \frac{f_X(x)}{q_X(x)} \right) f_X(x) dx \tag{124}$$

The importance sampling estimator will reduce variance if this integral is positive, which will occur if we select an importance sampling density that that meets the following conditions:

- $\frac{f_X(x)}{q(X)} > 1$ when $h(x)f_X(x)$ is small; *and*
- $\frac{f_X(x)}{q(X)} < 1$ when $h(x)f_X(x)$ is large.

We can therefore define the region where $h(x)f_X(x)$ is large as the *important part* of the distribution because it makes the largest contribution to the variance in Eq. (122).

We therefore aim to design the importance sampling density such that it places *high density in the important region and low density in regions that do not contribute significantly to the variance.*

Optimal Importance Sampling Density

Using the intuition we gain from Eq. (124), let us consider that we select a density $q_X(x) \propto h(x)f_X(x)$. Note that $h(x)f_X(x)$ does not define a valid density because it does not integrate to 1. In fact $\int_{-\infty}^{\infty} h(x)f_X(x)dx = \mathbb{E}[h(X)] = \theta$. We therefore use the following importance sampling density

$$q_X(x) = \frac{h(x)f_X(x)}{\theta}, \tag{125}$$

which is a valid probability density. Applying this density and computing the variance according to Eq. (123) yields

$$\begin{aligned} \text{Var}_q(\hat{\theta}_{is}) &= \int_{-\infty}^{\infty} \frac{h^2(x)f_X(x)\theta}{h(x)f_X(x)} f_X(x)dx - \theta^2 \\ &= \int_{-\infty}^{\infty} \theta h(x)f_X(x)dx - \theta^2 \\ &= \theta^2 - \theta^2 \\ &= 0 \end{aligned} \tag{126}$$

This *importance sampling density* yields a **zero variance estimator**! This means that if we could somehow sample from this density, then we would obtain a perfect estimate of θ, which is the optimal estimator. For this reason, we refer to this choice as the *optimal importance sampling density*. Unfortunately, we quickly realize that the optimal importance sampling density depends on θ – the quantity we are trying to estimate. It is therefore impossible, in theory, to sample from this density. However, we may use this result to identify a good importance sampling density.

Identifying a Good Importance Sampling Density

Even if we cannot identify the integration constant θ, the optimal importance sampling density in Eq. (125), the insights we gain from this optimal solution and the integral in Eq. (124) suggest that we want to select an importance sampling density that is *similar* in shape to $h(x)f_X(x)$.

One common choice is to select $q_X(x)$ such that it takes a maximum value at the same value that maximizes $h(x)f_X(x)$. This is referred to as selecting the importance sampling density according to the *maximum principle*. Of course, there are an infinite number of probability densities that satisfy the maximum principle. Regardless of the choice, this is often enough to ensure a significant variance reduction. Two strategies are commonly applied here. The first strategy is to select $q_X(x)$ by simply shifting the original distribution $f_X(x)$ such that it takes maximum value at $h(x)f_X(x)$. Another common approach is to define $q_X(x)$ as a normal probability density with mean value equal at the maximum of $h(x)f_X(x)$ and variance equal to that of the original density $f_X(x)$.

Figure 7 shows a simple function $h(X) = e^x$ with input from the standard normal distribution $f_X(x)$. Also plotted are the optimal importance sampling density and a standard normal probability density function that satisfies the maximum principle.

Rare Event Simulation & Probability of Failure Estimation

Perhaps the most common application of importance sampling is for the simulation of rare events and Monte Carlo estimation of small probability events. Consider the following example.

We aim to estimate $\theta = P(X \geq 20)$ where $X \sim N(0, 1)$ using Monte Carlo simulation. We can repose this problem in the form of the following expectation

$$\theta = P(X \geq 20) = \mathbb{E}[I_{X \geq 20}] = \int_{-\infty}^{\infty} I_{X \geq 20}\phi(x)dx \tag{127}$$

where

$$I_{X \geq 20} = \begin{cases} 1 & \text{if } X \geq 20 \\ 0 & \text{otherwise} \end{cases} \tag{128}$$

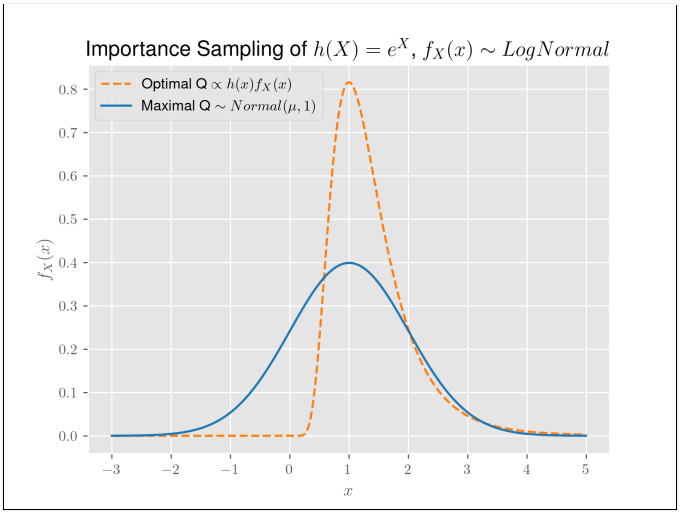


Figure 7: An example of importance sampling functions to estimate $\mathbb{E}[e^X]$.Uploaded new example with $h(x) = e^x$ and $f_X(x) \propto LogNormal$

We can try to solve this problem using standard Monte Carlo simulation, but practically it is impossible since we would need $N \approx 2.7 \times 10^{89}$ samples on average to obtain even a single non-zero value for $I_{X \geq 20}$. Any smaller value of N is almost guaranteed to return a probability of 0.

However, applying importance sampling in this case is straightforward. We could simply define $q_X(x) \sim N(20, 1)$ and approximately half of the samples we draw would satisfy $I_{X \geq 20}$, yielding a much better estimator.

We will discuss the application of importance sampling for rare event simulation and probability of failure estimation in the module on reliability analysis.

8.4 Stratified Sampling

Stratified sampling takes advantage of the ability to sample from conditional probabilities to reduce variance. Consider again that we aim to estimate $\theta = \mathbb{E}[Y] = \mathbb{E}[h(\boldsymbol{X})]$ where the random vector $\boldsymbol{X}$ has sample space Ω that can be divided into a collection of M disjoint subsets (strata) denoted $\Omega_k, k = 1, \ldots, M$ of known probability with

$$\begin{aligned} &\cup_{k=1}^{M} \Omega_k = \Omega \\ &\Omega_p \cap \Omega_q = \emptyset, \forall p \neq q \\ &P(\Omega_k) = p_k > 0 \\ &\sum_k p_k = 1 \end{aligned} \tag{129}$$

Using stratified sampling, we will then generate conditional samples of $Y|\boldsymbol{X} \in \Omega_k$. That is, we will compute $Y = h(\boldsymbol{X})$ based on samples drawn from $\boldsymbol{X}$ in a specified stratum Ω_k.

Before introducing the Monte Carlo estimator, let us first introduce the *law of total expectation*, which is the foundation for the stratified sampling estimator. Given a random variable Y having expectation $\mathbb{E}[Y]$ and any random variable I defined on the same probability space, then

$$\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|I]]. \tag{130}$$

That is, the expected value of Y is equal to the expectation of the conditional expectation of Y given I. As a special case, consider that I is a random variable that partitions the sample space Ω as in Eq. (129), then

$$\mathbb{E}[Y] = \sum_k \mathbb{E}[Y|I=k]P(I=k) = \sum_k p_k \mathbb{E}[Y|I=k] \tag{131}$$

To compute the Monte Carlo estimator, let us define the random variable I such that $I = k$ if $\boldsymbol{X} \in \Omega_k$ having probabilities $P(I=k) = p_k = P(\boldsymbol{X} \in \Omega_k)$. Next, we will express the estimator θ using the *law of total expectation* as

$$\begin{aligned} \theta = \mathbb{E}[Y] &= \mathbb{E}[\mathbb{E}[Y|I]] \\ &= \sum_{k=1}^{M} p_k \mathbb{E}[Y|I=k] \end{aligned} \tag{132}$$

Next, we will use standard Monte Carlo to estimate $\mathbb{E}[Y|I=k]$ using M_k samples drawn from the conditional distribution $f_X(x|I=k)$ with estimator given by

$$\mathbb{E}[Y|I=k] \approx \hat{\theta}_k = \frac{1}{M_k} \sum_{i=1}^{M_k} Y_i \tag{133}$$

Applying the conditional estimator in Eq. (133) to the total estimator in Eq. (132) yields the *stratified sampling* estimator given by

$$\theta \approx \hat{\theta}_s = \sum_{k=1}^{M} p_k \hat{\theta}_k = \sum_{k=1}^{M} \frac{p_k}{M_k} \sum_{i=1}^{M_k} Y_i \tag{134}$$

Variance Reduction with Stratified Sampling

To achieve a variance reduction we need to, once again, show that our stratified sampling estimator $\hat{\theta}_s$ has lower variance than our traditional Monte Carlos estimator. Let us start by computing the variance of our estimator $\hat{\theta}_s$ as follows

$$\begin{aligned} \text{Var}(\hat{\theta}_s) &= \text{Var}\left(\sum_{k=1}^{M} p_k \hat{\theta}_k\right) \\ &= \sum_{k=1}^{M} p_k^2 \text{Var}(\hat{\theta}_k) \\ &= \sum_{k=1}^{M} p_k^2 \frac{\sigma_k^2}{M_k} \end{aligned} \tag{135}$$

where the first step follows from simply applying the variance operator to the summation and the second step follow from the fact that $\hat{\theta}_k$) is the standard Monte Carlo estimator in Eq. (133) where σ_k^2 is the variance of stratum Ω_k.

To determine whether the variance of the stratified sampling estimator in Eq. (135) is less than the traditional Monte Carlo estimator, we first need to determine the number of samples, M_k, drawn from each stratum. Let's start by using a proportional allocation in which the number of samples in each stratum is determined by the size of the stratum. Here, we will set $M_k = p_k N$. Applying this proportional relation to our variance estimator, Eq. (135) simplifies as

$$\text{Var}(\hat{\theta}_s) = \frac{1}{N} \sum_{k=1}^{M} p_k \sigma_k^2 \tag{136}$$

Let us now compare this with our traditional Monte Carlo estimator as

$$\text{Var}(\hat{\theta}_N) - \text{Var}(\hat{\theta}_s) = \frac{1}{N}\sigma^2 - \frac{1}{N}\sum_{k=1}^{M} p_k \sigma_k^2. \tag{137}$$

To show a variance reduction, we therefore need to show that $\sigma^2 - \sum_{k=1}^{M} p_k \sigma_k^2 > 0$ or $\sum_{k=1}^{M} p_k \sigma_k^2 < \sigma^2$. To do this, we will introduce the *law of total variance* or the *conditional variance formula*, which states that

$$\text{Var}(Y) = \mathbb{E}[\text{Var}(Y|I)] + \text{Var}(\mathbb{E}[Y|I]) \tag{138}$$

We therefore recognize that

$$\sigma^2 = \text{Var}(Y) \geq \mathbb{E}[\text{Var}(Y|I)] = \sum_{k=1}^{M} p_k \sigma_k^2, \tag{139}$$

thus proving that the stratified sampling estimator provides a variance reduction. We note, however, that the proportional allocation is not optimal and therefore greater variance reduction can be achieved. We explore this next.

Optimal Stratified Estimator

There are potentially two means of optimizing the stratified sampling estimator $\hat{\theta}_s$. The first is to optimize the design of the strata. That is, we can potentially define the strata Ω_k such that, when we sample from these strata, the variance is minimized for a given fixed number of samples. To this end, Shields et al. [28] showed that refining existing strata will always reduce variance compared with adding samples to existing strata. However, the determination of an optimal stratification remains an open area of research and therefore will not be explored in further detail here. A further discussion of this approach can be found in [29].

The second approach is to optimize the allocation of samples for a given stratification, again defined by Eq. (129). We have already seen the case of proportional allocation in which the number of samples is proportional to the size of each stratum. We mentioned that this allocation is not optimal. Here, we will show how to determine the optimal allocation.

We have previously seen that the variance of the general stratified sampling estimator $\hat{\theta}_s$ is given by Eq. (135). To minimize the variance, we need to solve the following constrained optimization problem:

$$\begin{aligned} &\min_{M_k} \sum_{k=1}^{M} p_k^2 \frac{\sigma_k^2}{M_k} \\ &\text{subject to} \quad \sum_{k=1}^{M} M_k = N \end{aligned} \tag{140}$$

This minimization can be solved exactly using the Method of Lagrange Multipliers yielding the following optimal solution

$$M_k^* = \left(\frac{p_k \sigma_k}{\sum_{k=1}^{M} p_k \sigma_k} \right) N. \tag{141}$$

Apply this optimal value of M_k results in an estimator with variance given by

$$\frac{\left(\sum_{k=1}^{M} p_k \sigma_k\right)^2}{N}. \tag{142}$$

The optimal allocation given in Eq. (141) make sense intuitively. If consider that all the stratum variance are equal to a constant, i.e. $\sigma_k = s \ \forall k$ then the optimal allocation is the proportional allocation. That

is, the optimal allocation suggests that more samples should be placed in larger strata. On the other hand, if all strata are equal size, i.e. $p_k = p \; \forall k$ then the optimal allocation is in proportion to the standard deviations of each stratum. Those with higher standard deviation should be allocated more samples. Thus, when allowing both p_k and σ_k to vary, the optimal allocation will favor large strata with high variance. This is analogous to importance sampling in the sense that we want to sample more heavily from the "important regions" – that is the regions with highest contribution to the variance.

Although stratified sampling can result in substantial variance reduction, the main challenge in applying the optimal sample allocation is that the stratum standard deviations, σ_k are not typically known. Therefore, they must be estimated in the process of performing Monte Carlo simulation. Often this is undesirable and a simple proportional allocation is made, knowing that this at least provide a variance reduction.

8.5 Latin Hypercube Sampling

Latin hypercube sampling is a special variant of stratified sampling in which range of the individual components, X_i of the random vector $\boldsymbol{X} = [X_1, X_2, \ldots, X_n]^T$ are divided (stratified) into N disjoint subsets of *equal probability*. Random samples of each component are then drawn from each of the strata and are randomly paired to assemble samples of the random vector. Latin hypercube sampling is perhaps the most widely used variance reduction method because it is both very easy to implement and produces a substantial variance reduction for many problems.

Latin hypercube sampling is conducted in the following general steps:

1. Divide the range of each random variable into N strata of equal probability
2. Draw a single sample from each stratum
3. Randomly pair the samples without replacment

We notice that, because we pair the samples randomly that a Latin hypercube sample is not unique. There are many ways in which the samples drawn from each variable can be combined. Extensive studies have been performed to identify different means of combining the samples from different variables to achieve different objectives such as enhancing the space-filling of the sample set and reducing spurious correlations that may existing in the sample set. We will not discuss these here.

The variance reduction for Latin hypercube sampling is more difficult to understand. To understand the variance reduction, we need to envision the sample domain as a set of N^n cells of equal probability $\frac{1}{N^n}$ defined by the intersection of the strata in each component. We can label these cells with a indices $r = 1, 2, \ldots, N^n$ and assign each cell a set of coordinates $\mathbf{m}_r$ corresponding to the indices of the intervals in each coordinate as

$$\mathbf{m}_r = (m_{r1}, m_{r2}, \ldots, m_{rn}) \tag{143}$$

This is illustrated in two-dimensions in Figure 8. [Make this figure] The variance of the Latin hypercube sampling estimator $\hat{\theta}_l$ can then be given relative to the standard Monte Carlo estimator as

$$\mathrm{Var}(\hat{\theta}_l) = \mathrm{Var}(\hat{\theta}_l) + \frac{N-1}{N} \frac{1}{N^n (N-1)^n} \sum_{\mathcal{R}} (\mu_p - \theta)(\mu_q - \theta) \tag{144}$$

where μ_p is the mean value of Latin hypercube cell p and $\mathcal{R}$ is the restricted set of all pairs (μ_p, μ_q) corresponding to cells having no cell coordinates in common, i.e. $m_{rj} \neq m_{sj} \forall j$. An example of these pairs of cells is illustrated, again in two dimensions, in Figure 8.

This variance relation was first derived by McKay et al. [30] who further stated the following theorem.

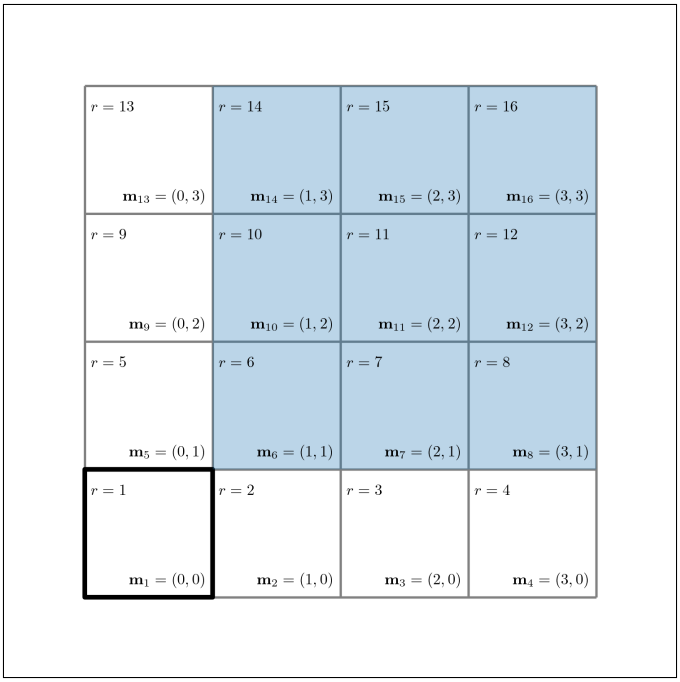


Figure 8: An example the first cell in a Latin Hypercube, shown in bold at $r = 1$. This cell may be associated with any other cell that does not share a coordinate, shaded in blue.Uploaded new figure with edits after September 7 advisee meeting

Theorem 2 *If $y = f(x_1, \ldots, x_n)$ is monotonic in each of the x_j, then*

$$Var(\hat{\theta}_l) \leq Var(\hat{\theta}_n) \tag{145}$$

Many subsequent investigations have further clarified and provided insights into the variance reduction properties of Latin hypercube sampling. Perhaps most importantly (and intuitively) it was shown by Stein [31] that Latin hypercube sampling has the effect of filtering our the variance associated with the additive components of the function, i.e. the part of the function that depends only on individual variables and does not involve the interaction of variables. Consider that the function $y = f(\boldsymbol{x})$ can be expressed as

$$f(\boldsymbol{x}) = f_a(\boldsymbol{x}) + r(\boldsymbol{x}) \tag{146}$$

where $f_a(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(x_i)$ is the best additive fit to the function and $r(\boldsymbol{x})$ contains the part of the function involving variable interactions. Stein showed that Latin hypercube reduces the variance associated with $f_a(\boldsymbol{x})$ and does no worse than standard Monte Carlo with random sampling on $r(\boldsymbol{x})$. Therefore, the closer the function is to additive, the better Latin hypercube sampling will perform as a variance reduction method.

References

[1] A. M. Law, W. D. Kelton, W. D. Kelton, Simulation modeling and analysis, Vol. 3, Mcgraw-hill New York, 2007.

[2] G. O. Roberts, W. R. Gilks, A. Gelman, Weak convergence and optimal scaling of random walk metropolis algorithms, The annals of applied probability 7 (1) (1997) 110–120.

[3] J. S. Rosenthal, et al., Optimal proposal distributions and adaptive mcmc, Handbook of Markov Chain Monte Carlo 4 (10.1201) (2011).

[4] S. Brooks, A. Gelman, G. Jones, X.-L. Meng, Handbook of markov chain monte carlo, CRC press, 2011.

[5] G. O. Roberts, J. S. Rosenthal, Coupling and ergodicity of adaptive markov chain monte carlo algorithms, Journal of applied probability 44 (2) (2007) 458–475.

[6] H. Haario, E. Saksman, J. Tamminen, An adaptive metropolis algorithm, Bernoulli (2001) 223–242.

[7] G. O. Roberts, J. S. Rosenthal, Examples of adaptive mcmc, Journal of computational and graphical statistics 18 (2) (2009) 349–367.

[8] L. Tierney, A. Mira, Some adaptive monte carlo methods for bayesian inference, Statistics in medicine 18 (17-18) (1999) 2507–2515.

[9] A. Mira, et al., On metropolis-hastings algorithms with delayed rejection, Metron 59 (3-4) (2001) 231–241.

[10] P. J. Green, A. Mira, Delayed rejection in reversible jump metropolis–hastings, Biometrika 88 (4) (2001) 1035–1053.

[11] H. Haario, M. Laine, A. Mira, E. Saksman, Dram: efficient adaptive mcmc, Statistics and computing 16 (2006) 339–354.

[12] J. A. Vrugt, H. V. Gupta, W. Bouten, S. Sorooshian, A shuffled complex evolution metropolis algorithm for optimization and uncertainty assessment of hydrologic model parameters, Water resources research 39 (8) (2003).

[13] C. J. T. Braak, A markov chain monte carlo version of the genetic algorithm differential evolution: easy bayesian computing for real parameter spaces, Statistics and Computing 16 (2006) 239–249.

[14] J. A. Vrugt, C. Ter Braak, C. Diks, B. A. Robinson, J. M. Hyman, D. Higdon, Accelerating markov chain monte carlo simulation by differential evolution with self-adaptive randomized subspace sampling, International journal of nonlinear sciences and numerical simulation 10 (3) (2009) 273–290.

[15] J. A. Vrugt, Markov chain monte carlo simulation using the dream software package: Theory, concepts, and matlab implementation, Environmental Modelling & Software 75 (2016) 273–316.

[16] J. Goodman, J. Weare, Ensemble samplers with affine invariance, Communications in applied mathematics and computational science 5 (1) (2010) 65–80.

[17] C. J. Geyer, Practical markov chain monte carlo, Statistical science (1992) 473–483.

[18] C. J. Geyer, Introduction to markov chain monte carlo, Handbook of markov chain monte carlo 20116022 (2011) 45.

[19] A. Gelman, D. B. Rubin, Inference from iterative simulation using multiple sequences, Statistical science 7 (4) (1992) 457–472.

[20] J. M. Flegal, M. Haran, G. L. Jones, Markov chain monte carlo: Can we trust the third significant figure?, Statistical Science (2008) 250–260.

[21] J. Besag, Comments on "representations of knowledge in complex systems" by u. grenander and mi miller, J. Roy. Statist. Soc. Ser. B 56 (591-592) (1994) 4.

[22] G. O. Roberts, R. L. Tweedie, Exponential convergence of langevin distributions and their discrete approximations, Bernoulli (1996) 341–363.

[23] G. O. Roberts, J. S. Rosenthal, Optimal scaling of discrete approximations to langevin diffusions, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 60 (1) (1998) 255–268.

[24] R. M. Neal, Mcmc using hamiltonian dynamics, in: S. Brooks, A. Gelman, G. Jones, X.-L. Meng (Eds.), Handbook of markov chain monte carlo, Chapman and Hall/CRC, New York, 2011, Ch. 5, pp. 113–160.

[25] M. Betancourt, A conceptual introduction to hamiltonian monte carlo (2018). `arXiv:1701.02434`.

[26] M. D. Hoffman, A. Gelman, et al., The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo., J. Mach. Learn. Res. 15 (1) (2014) 1593–1623.

[27] S. M. Ross, Introduction to probability models, Academic press, 2014.

[28] M. D. Shields, K. Teferra, A. Hapij, R. P. Daddazio, Refined stratified sampling for efficient monte carlo based uncertainty quantification, Reliability Engineering & System Safety 142 (2015) 310–325.

[29] M. D. Shields, Adaptive monte carlo analysis for strongly nonlinear stochastic systems, Reliability Engineering & System Safety 175 (2018) 207–224.

[30] M. D. McKay, R. J. Beckman, W. J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 42 (1) (2000) 55–61.

[31] M. Stein, Large sample properties of simulations using latin hypercube sampling, Technometrics 29 (2) (1987) 143–151.

Nomenclature

Functions

$P(\cdot)$ Probability measure

$f_X(\cdot)$ Probability density function (PDF) of a random variable X

$F_X(\cdot)$ Cumulative distribution function (CDF) of a random variable X

$Q(\cdot)$ Quantile function, defined as the inverse of the cumulative distribution function, $F_X^{-1}(x)$

$\Phi(\cdot)$ Cumulative distribution function of a standard normal random variable

$\mathbb{E}[\cdot]$ Expected value of a random variable. Also denoted $\mu_X \triangleq \mathbb{E}[X]$

$\mathbb{E}[X^n]$ The n^{th} moment about the origin of a random variable X

$\mathrm{Var}(\cdot)$ Variance of the random variable. Also denoted $\sigma_X^2 \triangleq \mathrm{Var(X)}$

$\mathrm{Cov}(\cdot)$ The covariance of the random variable input

Operators

$\cap$ Intersection

$\cup$ Union

Variables

X A random variable

$\boldsymbol{X}$ A random vector in $\mathbb{R}^n$

Ω Sample space

Ω_k Strata within the sample space Ω

$\hat{\theta}$ Generic estimator for the quantity θ