CHAPTER

# NUMERICAL APPROXIMATION OF PREDICTIVE DISTRIBUTION 19

**CHAPTER CONTENTS**

MAP estimation explained in Section 17.3 is a useful alternative to MLE due to its simplicity and ability to mitigate overfitting. However, since only a single parameter value,

$$\widehat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \underset{\boldsymbol{\theta}}{\mathrm{argmax}}\, p(\boldsymbol{\theta}|\mathcal{D}),$$

is used, the distinctive feature of Bayesian inference that infinitely many parameters are considered is lost. Therefore, the obtained density estimator is always included in the parametric model (Fig. 17.1). In this chapter, algorithms for numerically approximating the Bayesian predictive distribution,

$$\widehat{p}_{\mathrm{Bayes}}(\boldsymbol{x}) = \int q(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})\mathrm{d}\boldsymbol{\theta}, \tag{19.1}$$

are introduced.

## 19.1 MONTE CARLO INTEGRATION

The *Monte Carlo method* is a generic name of algorithms that use random numbers, and its name stems from the Monte Carlo Casino in Monaco. In this section, the method of *Monte Carlo integration* is introduced to numerically approximate the integration in Eq. (19.1).

More specifically, Monte Carlo integration approximates the expectation of a function $g(\boldsymbol{\theta})$,

$$\int g(\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \tag{19.2}$$

**An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00030-3**

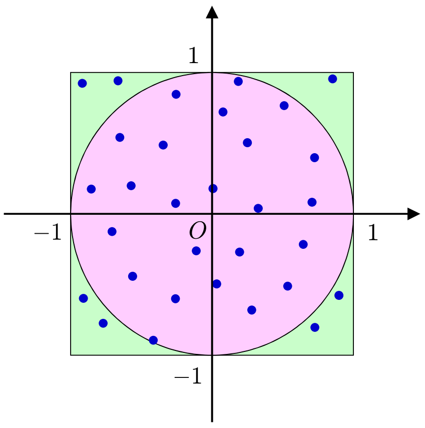


**FIGURE 19.1**

Numerical computation of $\pi$ by Monte Carlo integration.

by the average over i.i.d. samples $\{\boldsymbol{\theta}_i\}_{i=1}^n$ following $p(\boldsymbol{\theta})$ as

$$\frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{\theta}_i). \tag{19.3}$$

By the law of large numbers (see Section 7.3), consistency of Monte Carlo integration is guaranteed. Namely, in the limit $n \to \infty$, Eq. (19.3) converges in probability to Eq. (19.2).

As illustration, let us approximate $\pi \approx 3.14$ by Monte Carlo integration. Let us consider a $2 \times 2$ square and its inscribed circle of radius 1 (Fig. 19.1). Let $g(x, y)$ be the function defined by

$$g(x,y) = \begin{cases} 1 & x^2 + y^2 \leq 1, \\ 0 & \text{otherwise,} \end{cases}$$

and let $p(x, y)$ be the uniform distribution on $[-1, 1]^2$:

$$p(x,y) = \begin{cases} 1/4 & -1 \leq x, y \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Then the area of the inscribed circle, which is equal to $\pi$, is expressed as

$$\pi = 4 \iint g(x,y)p(x,y)\mathrm{d}x\mathrm{d}y.$$

```
n=10000000; x=rand(n,2)*2-1; pih=4*mean(sum(x.^2,2)<=1)
```

**FIGURE 19.2**

MATLAB code for numerically computing $\pi$ by Monte Carlo integration.

Let us draw $n$ i.i.d. samples $\{(x_i, y_i)\}_{i=1}^{n}$ following the uniform distribution $p(x, y)$ and compute the above expectation by Monte Carlo integration:

$$\pi \approx \frac{4}{n}\sum_{i=1}^{n} g(x_i, y_i) = \frac{4n'}{n},$$

where $n'$ denotes the number of samples included in the circle. In the limit $n \to \infty$, $4n'/n$ converges in probability to $\pi$.

A MATLAB code for numerically computing $\pi$ by Monte Carlo integration is given in Fig. 19.2, which tends to give 3.14.

## 19.2 IMPORTANCE SAMPLING

To perform Monte Carlo integration for approximating the Bayesian predictive distribution given by Eq. (19.1), random samples need to be generated following the posterior probability $p(\boldsymbol{\theta}|\mathcal{D})$. Techniques to generate random samples from an arbitrary probability distribution will be discussed in Section 19.3. In this section, another approach called *importance sampling* is introduced, which approximates the expectation over any (complicated) target probability density $p(\boldsymbol{\theta})$ based on another (simple) probability density $p'(\boldsymbol{\theta})$ such as the normal distribution. $p'(\boldsymbol{\theta})$ is called a *proxy distribution*.

In importance sampling, samples $\{\boldsymbol{\theta}'_{i'}\}_{i'=1}^{n'}$ drawn i.i.d. from $p'(\boldsymbol{\theta})$ are first generated. Then the expectation of a function $g(\boldsymbol{\theta})$ over any $p(\boldsymbol{\theta})$ is approximated by the average over $\{\boldsymbol{\theta}'_{i'}\}_{i'=1}^{n'}$ weighted according to the *importance* $p(\boldsymbol{\theta})/p'(\boldsymbol{\theta})$:

$$\int g(\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} = \int \left(g(\boldsymbol{\theta})\frac{p(\boldsymbol{\theta})}{p'(\boldsymbol{\theta})}\right)p'(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} \approx \frac{1}{n'}\sum_{i'=1}^{n'} g(\boldsymbol{\theta}_{i'})\frac{p(\boldsymbol{\theta}_{i'})}{p'(\boldsymbol{\theta}_{i'})}.$$

As the name stands for, $p(\boldsymbol{\theta})/p'(\boldsymbol{\theta})$ indicates how important a sample drawn from $p'(\boldsymbol{\theta})$ is in $p(\boldsymbol{\theta})$. By the law of large numbers (see Section 7.3), consistency of importance sampling is guaranteed. Namely, in the limit $n \to \infty$, the importance-weighted average $\frac{1}{n'}\sum_{i'=1}^{n'} g(\boldsymbol{\theta}_{i'})\frac{p(\boldsymbol{\theta}_{i'})}{p'(\boldsymbol{\theta}_{i'})}$ converges in probability to the true expectation $\int g(\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}$.

Let us compute the expectation of function $g(\theta) = \theta^2$ over the standard *Laplace distribution* (see Section 4.5) by importance sampling using the standard normal distribution as a proxy:

```
n=10000000; s=3; x=randn(n,1)*s; x2=x.^2; ss=2*s^2;
t=mean(x2.*(exp(-abs(x))/2)./(exp(-x2./ss)/sqrt(ss*pi)))
```

**FIGURE 19.3**

MATLAB code for importance sampling.

$$\int \theta^2 p(\theta)\mathrm{d}\theta \approx \frac{1}{n'}\sum_{i'=1}^{n'} \theta_{i'}^2 \frac{p(\theta_{i'})}{p'(\theta_{i'})}, \tag{19.4}$$

where $p(\theta)$ and $p'(\theta)$ are the standard Laplace and Gaussian densities:

$$p(\theta) = \frac{1}{2}\exp(-|\theta|) \quad \text{and} \quad p'(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\theta^2}{2\sigma^2}\right).$$

Since the left-hand side of Eq. (19.4) is actually the variance of the standard Laplace distribution, La(0, 1), its true value is 2. A MATLAB code for computing the right-hand side of Eq. (19.4) is given in Fig. 19.3, which tends to give 2.00.

Although importance sampling is guaranteed to be consistent, its variance can be large depending on the choice of a proxy distribution. This implies that a large number of samples may be needed to obtain a reliable value by importance sampling.

## 19.3 SAMPLING ALGORITHMS

In this section, methods for generating random samples from an arbitrary probability distribution are introduced, based on a random sample generator for the uniform distribution or the normal distribution (e.g. the `rand` or the `randn` functions in MATLAB). By directly generating i.i.d. samples $\{\boldsymbol{\theta}_i\}_{i=1}^n$ from $p(\boldsymbol{\theta})$, the expectation of a function $g(\boldsymbol{\theta})$ can be approximated as

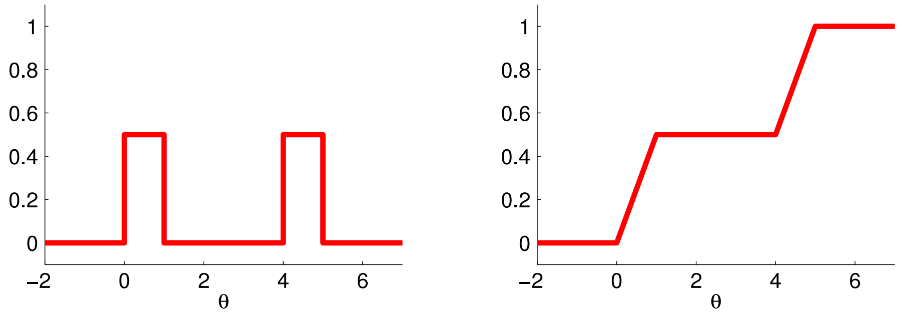
$$\int g(\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} \approx \frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{\theta}_i).$$

### 19.3.1 INVERSE TRANSFORM SAMPLING

*Inverse transform sampling* generates a one-dimensional random sample $\theta$ that follows a probability distribution with density $p(\theta)$ based on a uniform random variable $u$ on $[0,1]$ [62] and the *cumulative distribution function* of $p(\theta)$. The cumulative distribution function of $p(\theta)$, denoted by $P(\theta)$, is defined as follows (Fig. 19.4):

$$P(\theta) = \int_{-\infty}^{\theta} p(u)\mathrm{d}u.$$

Let $\theta = P^{-1}(u)$ be the *inverse function* of $u = P(\theta)$. Then, for the uniform random variable $u$ on $[0,1]$, $\theta = P^{-1}(u)$ has probability density $p(\theta)$ (Fig. 19.5). Thus, for $n$



(a) Probability density function $p(\theta)$ (b) Cumulative distribution function $P(\theta)$

**FIGURE 19.4**

Examples of probability density function $p(\theta)$ and its cumulative distribution function $P(\theta)$. Cumulative distribution function is monotone nondecreasing and satisfies $\lim_{\theta\to-\infty} P(\theta) = 0$ and $\lim_{\theta\to\infty} P(\theta) = 1$.

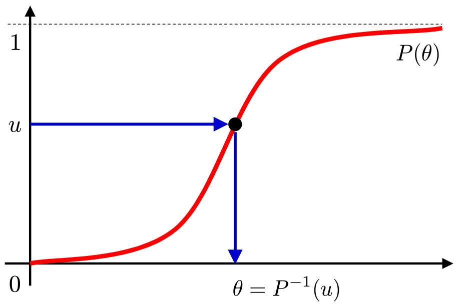


**FIGURE 19.5**

Inverse transform sampling.

uniform random variables $\{u_i\}_{i=1}^n$ on $[0, 1]$,

$$\{\theta_i \mid \theta_i = P^{-1}(u_i)\}_{i=1}^n$$

are i.i.d. with $p(\theta)$.

The validity of the above algorithm can be proved as follows. Since $\theta = P^{-1}(u)$, for any $\tau$,

$$\Pr(\theta \le \tau) = \Pr(P^{-1}(u) \le \tau).$$

As illustrated in Fig. 19.6, $\theta \le \theta'$ implies $P(\theta) \le P(\theta')$, and therefore

$$\Pr(P^{-1}(u) \le \tau) = \Pr(u \le P(\tau)).$$

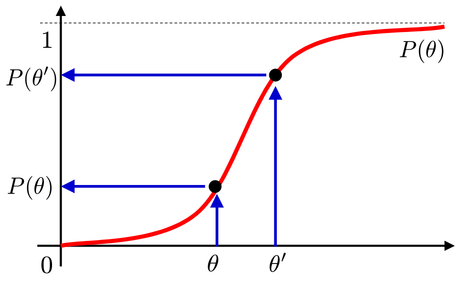


**FIGURE 19.6**

$\theta \leq \theta'$ implies $P(\theta) \leq P(\theta')$.

Furthermore, since $u$ follows the uniform distribution on $[0, 1]$,

$$\Pr(u \leq P(\tau)) = \int_0^{P(\tau)} \mathrm{d}u = P(\tau)$$

holds and therefore

$$\Pr(\theta \leq \tau) = P(\tau).$$

This means that the cumulative distribution function of $\theta$ generated by inverse transform sampling agrees with the target $P(\tau)$.

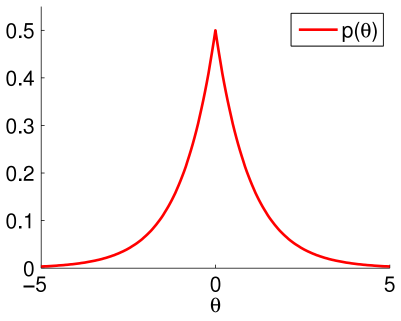
Let us generate random samples $\{\theta_i\}_{i=1}^n$ that are i.i.d. with the standard Laplace distribution $\mathrm{La}(0, 1)$. The probability density function $p(\theta)$, cumulative distribution function $P(\theta)$, and its inverse function $P^{-1}(u)$ are given as follows (see Fig. 19.7):

$$p(\theta) = \frac{1}{2}e^{-|\theta|}, \quad P(\theta) = \frac{1}{2}\left(1 + \mathrm{sign}(\theta)\left(1 - e^{-|\theta|}\right)\right),$$
$$P^{-1}(u) = -\mathrm{sign}\left(u - \frac{1}{2}\right)\log\left(1 - 2\left|u - \frac{1}{2}\right|\right),$$

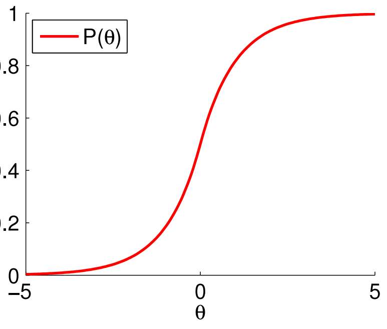
where $\mathrm{sign}(\theta)$ denotes the *sign function*:

$$\mathrm{sign}(\theta) = \begin{cases} 1 & (\theta > 0), \\ 0 & (\theta = 0), \\ -1 & (\theta < 0). \end{cases}$$

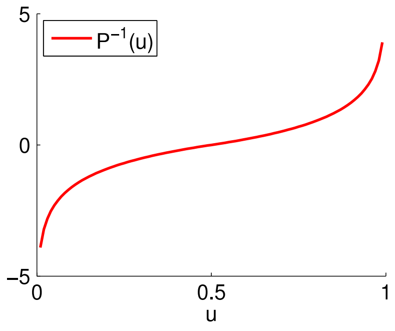
A MATLAB code for inverse transform sampling is given in Fig. 19.8, and its behavior is illustrated in Fig. 19.9.



(a) Probability density function $p(\theta)$



(b) Cumulative distribution function $P(\theta)$



(c) Inverse cumulative distribution function $P^{-1}(u)$

**FIGURE 19.7**

Laplace distribution.

```
n=10000; u=rand(1,n); y=-sign(u-1/2).*log(1-2*abs(u-1/2));
figure(1); clf; hist(u,50);
figure(2); clf; hist(y,linspace(-8,8,30));
```

**FIGURE 19.8**

MATLAB code for inverse transform sampling.

As shown above, inverse transform sampling is a simple algorithm to generate samples following an arbitrary distribution. However, it can be applied only to one-dimensional distributions. Furthermore, the inverse cumulative distribution function $\theta = P^{-1}(u)$ needs to be explicitly computed, which can be difficult depending on the probability distributions.

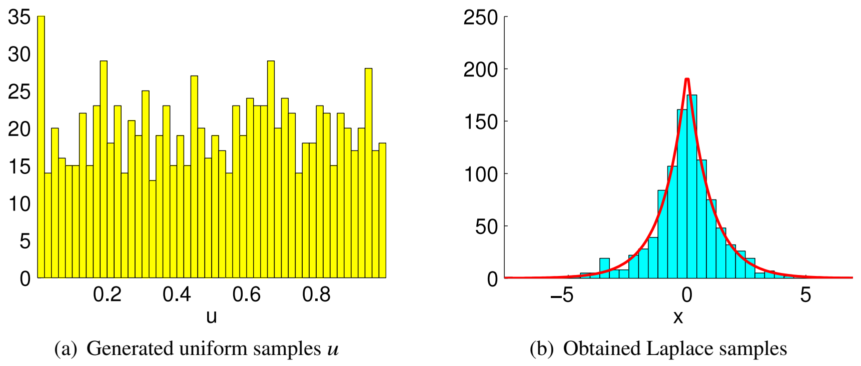


**FIGURE 19.9**

Example of inverse transform sampling for Laplace distribution.

For $i = 1, \ldots, n$, iterate the following steps:

1. Generate a proposal point $\boldsymbol{\theta}'$ following a proposal distribution $p'(\boldsymbol{\theta})$.
2. Generate a uniform random sample $v$ on $[0, \kappa]$.
3. If $v > p(\boldsymbol{\theta}')/p'(\boldsymbol{\theta}')$, reject the proposal point $\boldsymbol{\theta}'$ and go to 1.
4. Accept the proposal point $\boldsymbol{\theta}'$ and set $\boldsymbol{\theta}_i \longleftarrow \boldsymbol{\theta}'$.
5. Increase the index: $i \longleftarrow i + 1$.

**FIGURE 19.10**

Algorithm of rejection sampling.

### 19.3.2 REJECTION SAMPLING

*Rejection sampling* is a computer-intensive approach to generating random samples following $p(\boldsymbol{\theta})$ based on samples drawn i.i.d. from another density $p'(\boldsymbol{\theta})$ [117]. $p'(\boldsymbol{\theta})$ is called a *proposal distribution*, and a simple distribution such as the uniform distribution or the normal distribution is usually chosen.

The assumption required in rejection sampling is that an upper bound of the probability density ratio $p(\boldsymbol{\theta})/p'(\boldsymbol{\theta})$ exists and is known:

$$\max_{\boldsymbol{\theta}} \left[ \frac{p(\boldsymbol{\theta})}{p'(\boldsymbol{\theta})} \right] \leq \kappa < \infty.$$

In rejection sampling, a sample $\boldsymbol{\theta}'$ is generated following a proposal distribution $p'(\boldsymbol{\theta})$, which is called a *proposal point*. Then from the uniform distribution on $[0, \kappa]$,

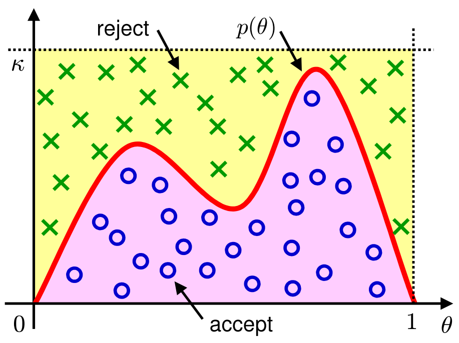


**FIGURE 19.11**

Illustration of rejection sampling when the proposal distribution is uniform.

a sample $v$ is generated, which is used for evaluating the proposal point $\boldsymbol{\theta}'$. If

$$v \leq \frac{p(\boldsymbol{\theta}')}{p'(\boldsymbol{\theta}')},$$

the proposal point $\boldsymbol{\theta}'$ is *accepted*, i.e. set $\boldsymbol{\theta}_i = \boldsymbol{\theta}'$ and $i \leftarrow i + 1$. Otherwise, the proposal point $\boldsymbol{\theta}'$ is *rejected*. This procedure is repeated until $n$ points are accepted. Then the accepted samples $\{\boldsymbol{\theta}_i\}_{i=1}^n$ have probability density $p(\boldsymbol{\theta})$. The algorithm of rejection sampling is summarized in Fig. 19.10.

The idea of rejection sampling can be more easily understood if $p(\boldsymbol{\theta})$ is defined on a finite set $\mathcal{X}$ (say $[0,1]$) and the uniform distribution on $\mathcal{X}$ is used as the proposal distribution $p'(\boldsymbol{\theta})$. As illustrated in Fig. 19.11, samples generated below/above the curve of $p(\boldsymbol{\theta})$ are accepted/rejected in this setup, which results in samples having probability density $p(\boldsymbol{\theta})$.

A MATLAB code for rejection sampling for the probability density function,

$$p(\theta) = \begin{cases} \frac{1}{4}\theta & (0 \leq \theta < 1), \\ \frac{1}{2} - \frac{1}{4}\theta & (1 \leq \theta < 2), \\ \frac{1}{4} & (2 \leq \theta < 3), \\ -\frac{3}{2} + \frac{1}{2}\theta & (3 \leq \theta < 4), \\ \frac{5}{2} - \frac{1}{2}\theta & (4 \leq \theta \leq 5), \end{cases}$$

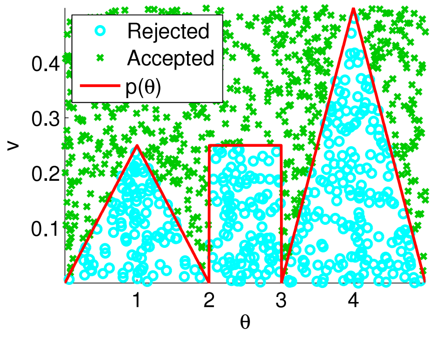
is given in Fig. 19.12, and its behavior is illustrated in Fig. 19.13.

Rejection sampling only requires the upper bound $\kappa$, while inverse transform sampling needs analytic computation of the inverse cumulative distribution function.

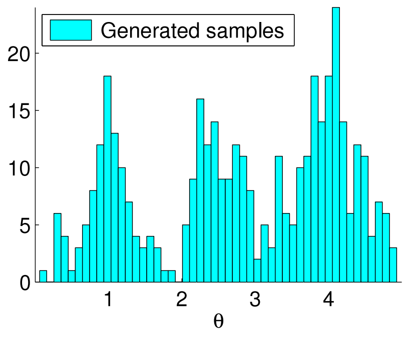
```
n=10000; u=5*rand(n,1); v=0.6*rand(n,1); y=zeros(n,1);
t=(0<=u & u<1); y(t)=0.25*u(t);
t=(1<=u & u<2); y(t)=-0.25*u(t)+0.5;
t=(2<=u & u<3); y(t)=0.25*ones(size(u(t)));
t=(3<=u & u<4); y(t)=0.5*u(t)-1.5;
t=(4<=u & u<=5); y(t)=-0.5*u(t)+2.5;
x=u(v<=y);
figure(1); clf; hold on; hist(x,50);
```

**FIGURE 19.12**

MATLAB code for rejection sampling.



(a) Generated uniform samples $(\theta, v)$



(b) Obtained samples following $p(\theta)$

**FIGURE 19.13**

Example of rejection sampling.

Thus, it is much easier to implement in practice. Furthermore, rejection sampling is immediately applicable to multidimensional probability distributions.

However, depending on the profile of $p(\boldsymbol{\theta})$, the probability of accepting proposal points can be small and then rejection sampling is computationally expensive. For example, when $p(\boldsymbol{\theta})$ has a sharp profile, $\kappa$ tends to be large and then the acceptance rate is small (see Fig. 19.14). Furthermore, depending on the choice of proposal distribution $p'(\boldsymbol{\theta})$, $\kappa$ can be infinity and then rejection sampling is not applicable.

## 19.3.3 MARKOV CHAIN MONTE CARLO (MCMC) METHOD

Rejection sampling cannot be used when the upper bound $\kappa$ is unknown, while MCMC methods do not have such a restriction. A *stochastic process* is a random

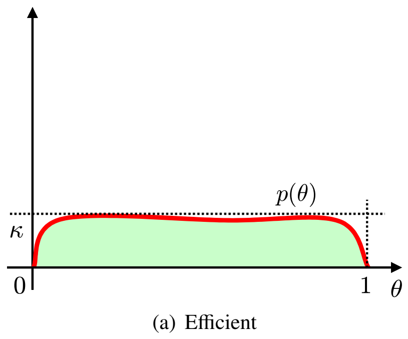


**FIGURE 19.14**

Computational efficiency of rejection sampling. (a) When the upper bound of the probability density, $\kappa$, is small, proposal points are almost always accepted and thus rejection sampling is computationally efficient. (b) When $\kappa$ is large, most of the proposal points will be rejected and thus rejection sampling is computationally expensive.

variable that changes over time, and a Markov chain $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ is a stochastic process where sample $\boldsymbol{\theta}_i$ at time $i$ depends only on the previous sample $\boldsymbol{\theta}_{i-1}$.

### Metropolis-Hastings sampling

*Metropolis-Hastings sampling* is a MCMC method [52, 72] that generates samples following a Markov chain. Thus, the proposal distribution to draw the next sample $\boldsymbol{\theta}_{i+1}$ depends on the current sample $\boldsymbol{\theta}_i$:

$$p'(\boldsymbol{\theta}|\boldsymbol{\theta}_i).$$

A proposal point $\boldsymbol{\theta}'$ generated from the above proposal distribution $p'(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$ is evaluated by a uniform random variable $v$ on $[0, 1]$. More specifically, if

$$v \leq \frac{p(\boldsymbol{\theta}')p'(\boldsymbol{\theta}_i|\boldsymbol{\theta}')}{p(\boldsymbol{\theta}_i)p'(\boldsymbol{\theta}'|\boldsymbol{\theta}_i)},$$

then the proposal point $\boldsymbol{\theta}'$ is accepted and set

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}'.$$

Otherwise, the proposal point $\boldsymbol{\theta}'$ is rejected and set

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i.$$

Note the difference from rejection sampling that the previous value is used when the proposal point is rejected. The initial value $\boldsymbol{\theta}_0$ needs to be chosen by a user.

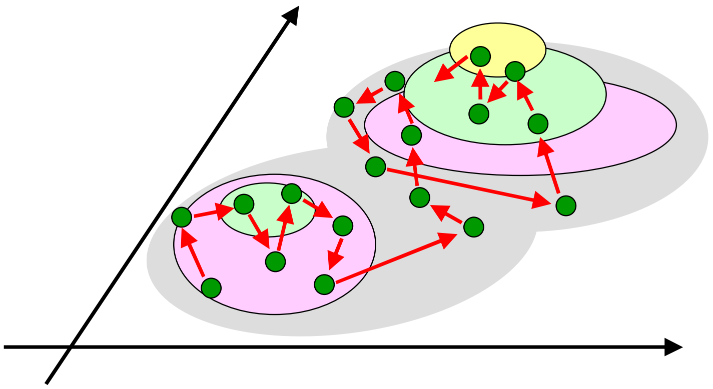


**FIGURE 19.15**

Random walk.

As a proposal distribution $p'(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$, the Gaussian distribution with expectation $\boldsymbol{\theta}_i$ and variance-covariance matrix $\sigma^2 \boldsymbol{I}_b$ may be used:

$$p'(\boldsymbol{\theta}|\boldsymbol{\theta}_i) = \frac{1}{(2\pi\sigma^2)^{b/2}} \exp\left(-\frac{(\boldsymbol{\theta}-\boldsymbol{\theta}_i)^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_i)}{2\sigma^2}\right),$$

where $b$ denotes the dimensionality of $\boldsymbol{\theta}$. Since the Gaussian distribution is symmetric, i.e.

$$p'(\boldsymbol{\theta}|\boldsymbol{\theta}_i) = p'(\boldsymbol{\theta}_i|\boldsymbol{\theta}),$$

the threshold for rejection is simplified as

$$\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta}_i)}.$$

When a local probability distribution such as the Gaussian distribution is used as the proposal distribution, the proposal point $\boldsymbol{\theta}'$ is generated around the previous parameter $\boldsymbol{\theta}_i$, and therefore points $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots$ move gradually. Such a random sequence is called a *random walk* (Fig. 19.15).

A MATLAB code for Metropolis-Hastings sampling is given in Fig. 19.16, and its behavior is illustrated in Fig. 19.17.

Unlike rejection sampling, Metropolis-Hastings sampling can be used without knowing the upper bound $\kappa$. Furthermore, even when the target probability density $p(\boldsymbol{\theta})$ is not explicitly available, Metropolis-Hastings sampling can still be employed, as long as $p(\boldsymbol{\theta})$ is known up to the *normalization term*. More specifically, suppose that $p(\boldsymbol{\theta})$ is unknown, but its unnormalized counterpart $\widetilde{p}(\boldsymbol{\theta})$ is available:

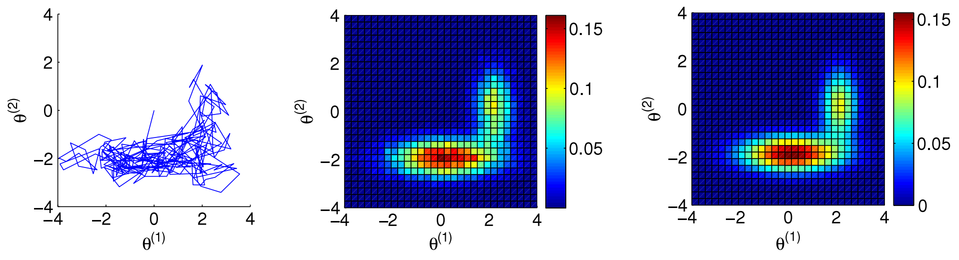
$$p(\boldsymbol{\theta}) = \frac{\widetilde{p}(\boldsymbol{\theta})}{Z},$$

```
n=100000; t=zeros(2,n);
for i=2:n
  u=randn(2,1)+t(:,i-1);
  if rand<=pdf(u)/pdf(t(:,i-1))
    t(:,i)=u;
  else
    t(:,i)=t(:,i-1);
  end
end

figure(1); clf; hold on
plot(t(1,1:500),t(2,1:500),'b-');

figure(2); clf; hold on
c=4; m=30; Q=zeros(m,m);
a=linspace(-c,c,m); b=-c+2*c/m*[1:m];
for k=1:n
  x=find(ceil(t(1,k)-b)==0,1);
  y=find(ceil(t(2,k)-b)==0,1);
  Q(x,y)=Q(x,y)+1;
end
surf(a,a,Q'/(2*c/m)^2/n);

figure(3); clf; hold on
P=zeros(m,m);
for X=1:m, for Y=1:m
  P(X,Y)=pdf([a(X);a(Y)]);
end, end
surf(a,a,P');
```

```
function p=pdf(x)

S1=[0.5 0; 0 4]; m1=x-[0; -2]; w1=0.7;
S2=[4 0; 0 1]; m2=x-[2; 0]; w2=0.3;
p=w1*sqrt(det(S1))/(2*pi)*exp(-m1'*S1*m1/2) ...
  +w2*sqrt(det(S2))/(2*pi)*exp(-m2'*S2*m2/2);
```

**FIGURE 19.16**

MATLAB code for Metropolis-Hastings sampling. The bottom function should be saved as "pdf.m."



(a) Trajectory of generated samples (only first 500 samples are plotted)
(b) Histogram of generated samples
(c) True probability density

**FIGURE 19.17**

Example of Metropolis-Hastings sampling.

where

$$Z = \int \widetilde{p}(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}$$

is the normalization constant which is unknown. Even in this situation, the rejection threshold can be computed as

$$\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta}_i)} = \frac{\widetilde{p}(\boldsymbol{\theta}')/Z}{\widetilde{p}(\boldsymbol{\theta}_i)/Z} = \frac{\widetilde{p}(\boldsymbol{\theta}')}{\widetilde{p}(\boldsymbol{\theta}_i)},$$

and therefore Metropolis-Hastings sampling can still be executed without any modification.

However, in the same way as rejection sampling, Metropolis-Hastings sampling is less efficient when the acceptance rate is low.

## *Gibbs sampling*

*Gibbs sampling* is another MCMC method that generates samples in an elementwise manner in multidimensional probability distributions [47].

More specifically, at time step $i$, for $j = 1, \ldots, d$, the proposal point of the $j$th dimension, $\theta_i^{(j)}$, is generated from the proposal distribution that depends on the previous sample $\boldsymbol{\theta}_{i-1} = (\theta_{i-1}^{(1)}, \ldots, \theta_{i-1}^{(d)})^\top$:

$$p(\theta^{(j)}|\theta_i^{(1)}, \ldots, \theta_i^{(j-1)}, \theta_{i-1}^{(j+1)}, \ldots, \theta_{i-1}^{(d)}).$$

Note that the above conditional density is different from

$$p(\theta^{(j)}|\theta_{i-1}^{(1)}, \ldots, \theta_{i-1}^{(j-1)}, \theta_{i-1}^{(j+1)}, \ldots, \theta_{i-1}^{(d)}),$$

```
n=100000; r=0.5; s=sqrt(1-r^2); t=zeros(2,n);
for i=2:n
  t(1,i)=s*randn+r*t(2,i-1); t(2,i)=s*randn+r*t(1,i);
end

figure(1); clf; hold on
u1=repmat(t(1,:),[2,1]); u1=u1(:);
u2=repmat(t(2,:),[2,1]); u2=u2(:);
plot(u1([2:200 200]),u2(1:200),'b-');
figure(2); clf; hold on
c=4; m=30; Q=zeros(m,m);
a=linspace(-c,c,m); b=-c+2*c/m*[1:m];
for k=1:n
  x=find(ceil(t(1,k)-b)==0,1);
  y=find(ceil(t(2,k)-b)==0,1);
  Q(x,y)=Q(x,y)+1;
end
surf(a,a,Q'/(2*c/m)^2/n);
figure(3); clf; hold on
P=zeros(m,m);
for X=1:m, for Y=1:m
  P(X,Y)=exp(-(a(X)^2-2*r*a(X)*a(Y)+a(Y)^2)/2/s)/(2*pi*s);
end, end
surf(a,a,P');
```

**FIGURE 19.18**

MATLAB code for Gibbs sampling.

i.e. not all values of the previous sample $\boldsymbol{\theta}_{i-1}$ are used, but the new values $\theta_i^{(1)}, \ldots, \theta_i^{(j-1)}$ are used up to the $(j-1)$th dimension.

Let us illustrate a more specific algorithm of Gibbs sampling for the two-dimensional Gaussian distribution:

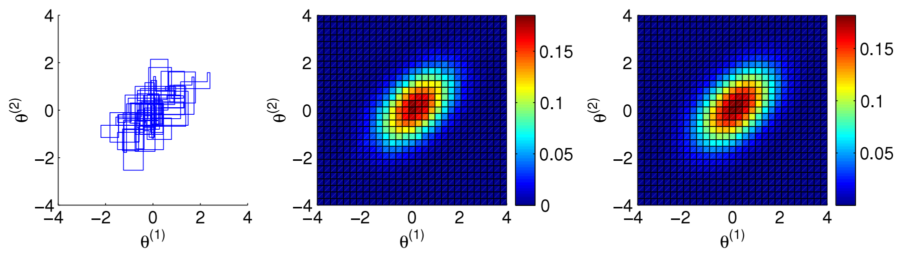
$$p(\boldsymbol{\theta}) = N(\boldsymbol{\theta}; \mathbf{0}_2, \boldsymbol{\Sigma}_\rho),$$

where, for $-1 < \rho < 1$,

$$\boldsymbol{\Sigma}_\rho = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}.$$

For $\boldsymbol{\theta} = (\theta^{(1)}, \theta^{(2)})^\top$, the conditional density is given by

$$p(\theta^{(1)}|\theta^{(2)}) = N(\theta^{(1)}; \rho\theta^{(2)}, 1-\rho^2).$$



(a) Trajectory of generated samples (only first 100 samples are plotted) (b) Histogram of generated samples (c) True probability density

**FIGURE 19.19**

Example of Gibbs sampling.

A MATLAB code of Gibbs sampling for the above Gaussian distribution is given in Fig. 19.18, and its behavior is illustrated in Fig. 19.19.

A variant of Gibbs sampling, which generates samples not only in a dimensionwise manner but also for some dimensions together, is called *blocked Gibbs sampling* [95]. Another variant that marginalizes some of the conditioning variables $\theta_i^{(1)},\ldots,\theta_i^{(j-1)},\theta_{i-1}^{(j+1)},\ldots,\theta_{i-1}^{(d)}$ is called *collapsed Gibbs sampling* [68].

Since Gibbs sampling does not involve proposal distributions and rejection, it is computationally efficient. However, it requires sampling from the conditional distribution, which may not always be straightforward in practice.

## Discussions

Samples $\{\boldsymbol{\theta}_i\}_{i=1}^n$ generated by rejection sampling are guaranteed to be mutually independent and have density $p(\boldsymbol{\theta})$ for finite $n$. On the other hand, samples $\{\boldsymbol{\theta}_i\}_{i=1}^n$ generated by MCMC methods are generally dependent on each other due to the incremental nature of Markov chains. Moreover, samples generated in an early stage may depend on the choice of initial value $\boldsymbol{\theta}_0$. For these reasons, samples generated by MCMC methods have density $p(\boldsymbol{\theta})$ only in the limit $n \to \infty$.

To mitigate these problems, it is common to discard samples generated in an early stage to decrease the dependency on the initial value (which is often referred to as *burn-in*) and adopt samples of only every $m$ time steps to weaken mutual dependency between samples.