STAT 576 Bayesian Analysis

Lecture 8: Markov Chain Simulation

Chencheng Cai

Washington State University

Backgrounds on Markov Chain

Let $\{X_t\}$ be a sequence of random variables. We say $\{X_t\}$ is a **Markov chain** if

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1} | X_t = x_t).$$

In other words, the future state of the chain depends only on the current state, not on the past states.

Backgrounds on Markov Chain

Let $\{X_t\}$ be a sequence of random variables. We say $\{X_t\}$ is a **Markov chain** if

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1} | X_t = x_t).$$

In other words, the future state of the chain depends only on the current state, not on the past states.

Remarks:

- If $t \in \mathbb{Z}^+$, then $\{X_t\}$ is a **discrete-time** Markov chain.
- If $t \in \mathbb{R}^+$, then $\{X_t\}$ is a **continuous-time** Markov chain.

Backgrounds on Markov Chain

Let $\{X_t\}$ be a sequence of random variables. We say $\{X_t\}$ is a **Markov chain** if

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1} | X_t = x_t).$$

In other words, the future state of the chain depends only on the current state, not on the past states.

Remarks:

- If $t \in \mathbb{Z}^+$, then $\{X_t\}$ is a **discrete-time** Markov chain.
- If $t \in \mathbb{R}^+$, then $\{X_t\}$ is a **continuous-time** Markov chain.
- For discrete-time Markov chain, we can define the **transition probability** $P_{ij,t} = P(X_{t+1} = j | X_t = i)$.
- For continuous-time Markov chain, we can define the **transition rate** $q_{ij,t} = \lim_{s \downarrow t} \frac{P(X_s = j | X_t = i) - \delta_{ij}}{s - t}$.

For now, we will focus on discrete-time Markov chain.

Backgrounds on Markov Chain

For discrete-time Markov chain with transition probability

$$P_{ij,t} = P(X_{t+1} = j | X_t = i),$$

- A Markov chain is **(time-)homogeneous** if the transition probabilities do not depend on t. Otherwise it is **(time-)inhomogeneous**.

Backgrounds on Markov Chain

For discrete-time Markov chain with transition probability

$$P_{ij,t} = P(X_{t+1} = j | X_t = i),$$

- A Markov chain is **(time-)homogeneous** if the transition probabilities do not depend on t. Otherwise it is **(time-)inhomogeneous**.
- The **state space** of the chain is the set of all possible values of X_t.

Backgrounds on Markov Chain

For discrete-time Markov chain with transition probability

$$P_{ij,t} = P(X_{t+1} = j | X_t = i),$$

- A Markov chain is **(time-)homogeneous** if the transition probabilities do not depend on t. Otherwise it is **(time-)inhomogeneous**.
- The **state space** of the chain is the set of all possible values of X_t.
- For a finite state space, we represent the transition probabilities in a matrix form, known as **the transition matrix**.

Backgrounds on Markov Chain

For discrete-time Markov chain with transition probability

$$P_{ij,t} = P(X_{t+1} = j | X_t = i),$$

- A Markov chain is **(time-)homogeneous** if the transition probabilities do not depend on t. Otherwise it is **(time-)inhomogeneous**.
- The **state space** of the chain is the set of all possible values of X_t.
- For a finite state space, we represent the transition probabilities in a matrix form, known as **the transition matrix**.
- For a measurable state space, we call it a **Markov chain on a measurable state space**. And we define the **transition kernel**

$$P(x, A) = P(X_{t+1} \in A | X_t = x), \quad x \in \mathcal{X}, A \in \mathcal{B}(\mathcal{X}).$$

For now, we focus on homogenoeous Markov chains with finite state space.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.
- A state i is **communicating** with state j if $i \to j$ and $j \to i$. We write $i \leftrightarrow j$.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.
- A state i is **communicating** with state j if $i \to j$ and $j \to i$. We write $i \leftrightarrow j$.
- A Markov chain is **irreducible** if any two states are communicating.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.
- A state i is **communicating** with state j if $i \to j$ and $j \to i$. We write $i \leftrightarrow j$.
- A Markov chain is **irreducible** if any two states are communicating.
- The **period** of a state i is the greatest common divisor of all $n \geq 1$ such that $P(X_n = i | X_0 = i) > 0$. If the period is 1, then the state is **aperiodic**.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.
- A state i is **communicating** with state j if $i \to j$ and $j \to i$. We write $i \leftrightarrow j$.
- A Markov chain is **irreducible** if any two states are communicating.
- The **period** of a state i is the greatest common divisor of all $n \geq 1$ such that $P(X_n = i | X_0 = i) > 0$. If the period is 1, then the state is **aperiodic**.
- A state i is **transient** if there is a non-zero probability that the chain will never return to i. Otherwise, it is **recurrent**.
- A state i is **positive-recurrent** if the expected return time to i is finite. Otherwise, it is **null-recurrent**.

Properties of Markov Chain

- A state i is **accessible** from state j if there exists $n \geq 0$ such that $P(X_n = i | X_0 = j) > 0$. We write $j \to i$.
- A state i is **communicating** with state j if $i \to j$ and $j \to i$. We write $i \leftrightarrow j$.
- A Markov chain is **irreducible** if any two states are communicating.
- The **period** of a state i is the greatest common divisor of all $n \geq 1$ such that $P(X_n = i | X_0 = i) > 0$. If the period is 1, then the state is **aperiodic**.
- A state i is **transient** if there is a non-zero probability that the chain will never return to i. Otherwise, it is **recurrent**.
- A state i is **positive-recurrent** if the expected return time to i is finite. Otherwise, it is **null-recurrent**.
- Periodicity, transience, recurrence and postive-recurrence are class properties.

Stationary Distribution of a Markov Chain

- A distribution π on the state space is **stationary** for a Markov chain with transition matrix P if

$$\pi = \pi P.$$

That is $\pi(i) = \sum_j \pi(j) P_{ji}$ for all i.

Stationary Distribution of a Markov Chain

- A distribution π on the state space is **stationary** for a Markov chain with transition matrix P if
$$\pi = \pi P.$$
That is $\pi(i) = \sum_j \pi(j) P_{ji}$ for all i.
- **Existence**:
Every positive recurrent Markov chain has a unique stationary distribution.
- **Uniquess**:
The stationary distribution is unique if the chain is irreducible.

Stationary Distribution of a Markov Chain

- A distribution π on the state space is **stationary** for a Markov chain with transition matrix P if
$$\pi = \pi P.$$
That is $\pi(i) = \sum_j \pi(j) P_{ji}$ for all i.
- **Existence**:
Every positive recurrent Markov chain has a unique stationary distribution.
- **Uniquess**:
The stationary distribution is unique if the chain is irreducible.
- **Covergence theorem**:
If the chain is irreducible and aperiodic with stationary π, then there exist constants $0 < \alpha < 1$ and $C > 0$ such that
$$\max_x \ \|P^t(x, \cdot) - \pi\|_{TV} \leq C\alpha^t.$$

Ergodic Theorem

- A state i is called **ergodic** if it is positive recurrent and aperiodic.

Ergodic Theorem

- A state i is called **ergodic** if it is positive recurrent and aperiodic.
- Let $V_i(n) = \sum_{t=0}^{n-1} I(X_t = i)$ be the number of visits to state i in the first n steps.

Ergodic Theorem

- A state i is called **ergodic** if it is positive recurrent and aperiodic.
- Let $V_i(n) = \sum_{t=0}^{n-1} I(X_t = i)$ be the number of visits to state i in the first n steps.
- **Ergodic Theorem**:
  For any irreducible and positive recurrent Markov chain with stationary distribution π, we have

$$\frac{V_i(n)}{n} \to \pi(i), \quad \text{a.s.}$$

Ergodic Theorem

- A state i is called **ergodic** if it is positive recurrent and aperiodic.
- Let $V_i(n) = \sum_{t=0}^{n-1} I(X_t = i)$ be the number of visits to state i in the first n steps.
- **Ergodic Theorem**:
  For any irreducible and positive recurrent Markov chain with stationary distribution π, we have
  $$\frac{V_i(n)}{n} \to \pi(i), \quad \text{a.s.}$$

Remarks:

- The convergence theorem indicates the marginal distribution at a future time is close to the stationary distribution (in terms of replications of the chain).
- The ergodic theorem indicates the partial sample from a long chain is close to the stationary distribution.

Detailed Balance

- A distribution π is **detailed balanced** for a Markov chain with transition matrix P if
$$\pi(i)P_{ij} = \pi(j)P_{ji}, \quad \forall i, j.$$
- A detailed balanced distribution is a stationary distribution. (Not vice versa!)

Detailed Balance

- A distribution π is **detailed balanced** for a Markov chain with transition matrix P if
$$\pi(i)P_{ij} = \pi(j)P_{ji}, \quad \forall i, j.$$
- A detailed balanced distribution is a stationary distribution. (Not vice versa!)
- Example: unique stationary distribution that is not detailed balanced.
Consider a Markov chain with state space $\{1, 2, 3\}$ and transition matrix
$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$
The stationary distribution is $\pi = (1/3, 1/3, 1/3)$, but it is not detailed balanced.

Markov Chain Simulation

- We generate a Markov chain with **designed** transition matrix P such that the stationary distribution of the chain is the target distribution.

Markov Chain Simulation

- We generate a Markov chain with **designed** transition matrix P such that the stationary distribution of the chain is the target distribution.
- We treat the values of the markov chain as samples from the target distribution.

Markov Chain Simulation

- We generate a Markov chain with **designed** transition matrix P such that the stationary distribution of the chain is the target distribution.
- We treat the values of the markov chain as samples from the target distribution.
- The chain is generated by the following steps:
  - Start from an initial state X_0.
  - At each step t, generate X_{t+1} from the conditional distribution $P(X_{t+1}|X_t)$.
  - Repeat the above step for n steps.
  - The samples $\{X_0, X_1, \ldots, X_n\}$ are the samples from the target distribution.

The Metropolis Algorithm

Assume the target distribution is $\pi(x)$. The Metropolis algorithm generates a Markov chain with the following steps:

The Metropolis Algorithm

Assume the target distribution is $\pi(x)$. The Metropolis algorithm generates a Markov chain with the following steps:

- Start from an initial state X_0.
- At each step t, generate a candidate state Y from a **symmetric proposal** distribution $q(x, y)$.
- Compute the acceptance probability

$$\alpha(x, y) = \min\left\{1, \frac{\pi(y)}{\pi(x)}\right\}.$$

- Generate X_{t+1} by

$$X_{t+1} = \begin{cases} Y & \text{with probability } \alpha(x, y), \\ X_t & \text{with probability } 1 - \alpha(x, y). \end{cases}$$

- Repeat the above step for n steps.

The Metropolis Algorithm

Justification on the stationary distribution:

- The transition probability from x to y is

$$P(x, y) = \min\left\{1, \frac{\pi(y)}{\pi(x)}\right\}$$

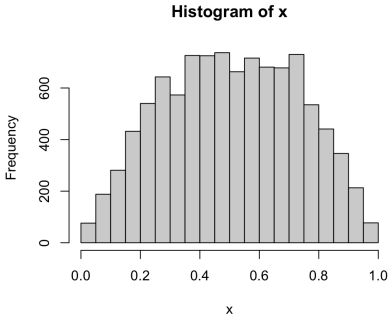
- The target distribution satisfies the detailed balance condition:

$$\pi(x)P(x, y) = \pi(y)P(y, x).$$

Implementation in R

Example: draw samples from $\mathrm{Beta}(2, 2)$ distribution.

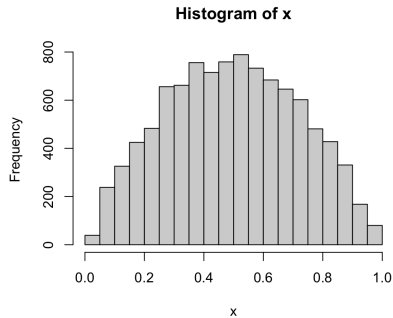
```
n = 10000
x = rep(0, n+1)
x[1] = 0.5
for(i in 1:n){
    y = x[i] + rnorm(1) * 0.5
    a = dbeta(y, 2, 2) / dbeta(x[i],
        2, 2)
    if (runif(1) <= a) x[i+1] = y
    else x[i+1] = x[i]
}
```



Implementation in R

Update: Use random starting point.

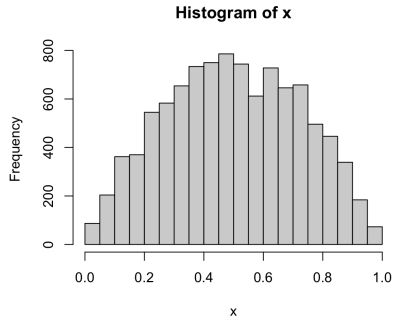
```
n = 10000
x = rep(0, n+1)
x[1] = runif(1)
for(i in 1:n){
    y = x[i] + rnorm(1) * 0.5
    a = dbeta(y, 2, 2) / dbeta(x[i],
        2, 2)
    if (runif(1) <= a) x[i+1] = y
    else x[i+1] = x[i]
}
```



Implementation in R

Update: Add burnin period to get samples under convergence.

```
n = 10000
burnin = 1000
x = rep(0, n+1)
x0 = 0.5
for (i in 1:burnin){
    y = x0 + rnorm(1) * 0.5
    a = dbeta(y, 2, 2) / dbeta(x0,
        2, 2)
    if (runif(1) <= a) x0 = y
}
x[1] = x0
for(i in 1:n){
    y = x[i] + rnorm(1) * 0.5
    a = dbeta(y, 2, 2) / dbeta(x[i],
         2, 2)
    if (runif(1) <= a) x[i+1] = y
    else x[i+1] = x[i]
}
```



Implementation in R

Update: Encapsulate the code into a function.

```
metropolis <- function(target, n, burnin, proposal, initial){
    sample = rep(0, n)
    x = initial()
    for(i in 1:(n+burnin)){
        y = proposal(x)
        a = target(y) / target(x)
        if(runif(1) <= a) x = y
        if(i>burnin)
            sample[i-burnin] = x
    }
    return(sample)
}

target = function(x){dbeta(x, 2, 2)}
proposal = function(x){x + rnorm(1)*0.5}
initial = function(){runif(1)}
s = metropolis(target, n, burnin, proposal, initial)
```

Implementation in R

Update: Use log density for higher precision.

```
metropolis.log <- function(log.target, n, burnin, proposal, initial){
    sample = rep(0, n)
    x = initial()
    for(i in 1:(n+burnin)){
        y = proposal(x)
        du = log.target(y) - log.target(x)
        if(runif(1) <= exp(du)) x = y
        if(i>burnin) sample[i-burnin] = x
    }
    return(sample)
}

log.target = function(x){dbeta(x, 2, 2, log=T)}
proposal = function(x){x + rnorm(1)*0.5}
initial = function(){runif(1)}
s = metropolis.log(log.target, n, burnin, proposal, initial)
```

Implementation in R

Update: Allow parallel sampling with multiple chains

```
metropolis <- function(log.target, n, burnin, proposal, initial, n.chain)
    {
    sample = array(dim=c(n.chain, n))
    x = initial(n.chain)
    for(i in 1:(n+burnin)){
        y = proposal(x)
        du = log.target(y) - log.target(x)
        accept = runif(n.chain) <= exp(du)
        x[accept] = y[accept]
        if(i>burnin) sample[,i-burnin] = x
    }
    return(sample)
}

log.target = function(x){dbeta(x, 2, 2, log=T)}
proposal = function(x){x + rnorm(length(x))*0.5}
initial = function(n){runif(n)}
s = metropolis(log.target, n, burnin, proposal, initial, 4)
```

Perofrmance of Simulating Mutliple Chain

```
for(n.chain in 0:6){
    start = Sys.time()
    for(i in 1:100)
        s = metropolis(log.target, n, burnin, proposal, initial,
                       2**n.chain)
    end = Sys.time()
    print(c(2 ** n.chain, (end-start)/100))
}
```

- Resutls on my laptop, M1 Pro (8-core CPU):

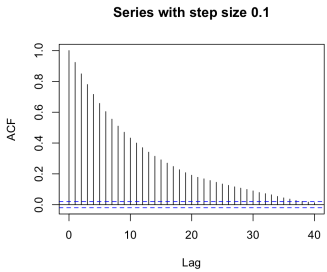
n.chain	1	2	4	8	16	32	64
time (ms)	53	58	65	79	107	173	298

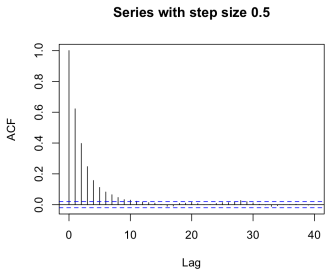
- Results on my desktop, 12900K (16-core CPU):

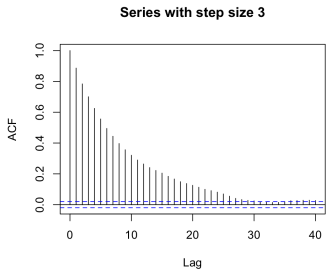
n.chain	1	2	4	8	16	32	64
time (ms)	34	40	44	52	69	106	176

Samples from the Metropolis Algorithm

Samples from the Metropolis algorithm are correlated.

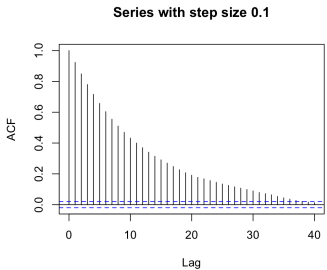


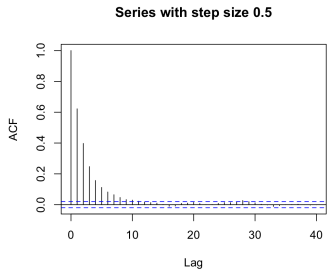


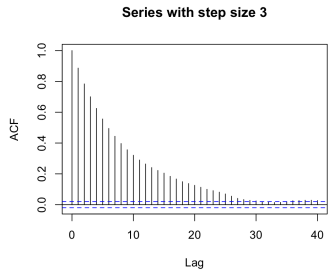


Samples from the Metropolis Algorithm

Samples from the Metropolis algorithm are correlated.







- A more local proposal distribution increases the autocorrelation.
- A more global proposal distribution resulting lower jumps increases the autocorrelation as well.
- One solution is to have asymmetric proposal distribution.

The Metropolis-Hastings Algorithm

Assume the target distribution is $\pi(x)$. The Metropolis-Hastings algorithm generates a Markov chain with the following steps:

The Metropolis-Hastings Algorithm

Assume the target distribution is $\pi(x)$. The Metropolis-Hastings algorithm generates a Markov chain with the following steps:

- Start from an initial state X_0.
- At each step t, generate a candidate state Y from a **proposal** distribution $q(x, y)$.
- Compute the acceptance probability

$$\alpha(x, y) = \min\left\{1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right\}.$$

- Generate X_{t+1} by

$$X_{t+1} = \begin{cases} Y & \text{with probability } \alpha(x, y), \\ X_t & \text{with probability } 1 - \alpha(x, y). \end{cases}$$

- Repeat the above step for n steps.

Implementation in R

```
metropolis.hastings <- function(log.target, n, burnin, proposal, initial,
     n.chain){
    sample = array(dim=c(n.chain, n))
    x = initial(n.chain)
    for(i in 1:(n+burnin)){
        prop = proposal(x)
        du = log.target(prop$y) - log.target(x)
        accept = runif(n.chain) <= exp(du+prop$dlog)
        x[accept] = prop$y[accept]
        if(i>burnin) sample[,i-burnin] = x
    }
    return(sample)
}
proposal = function(x){
    y = (0.5+x)/2 + rnorm(length(x))*0.5
    dlog = dnorm(x, (0.5+y)/2, 0.25, log=T) - dnorm(y, (0.5+x)/2, 0.25,
        log=T)
    return(list(y=y, dlog=dlog))
}
s = metropolis.hastings(log.target, n, burnin, proposal, initial, 1)
```

Comparison of Metropolis vs Metropolis-Hastings

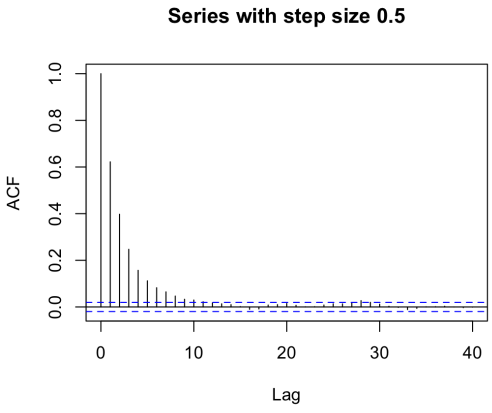


Figure: $q(x, y) = \mathcal{N}(x, 0.5^2)$

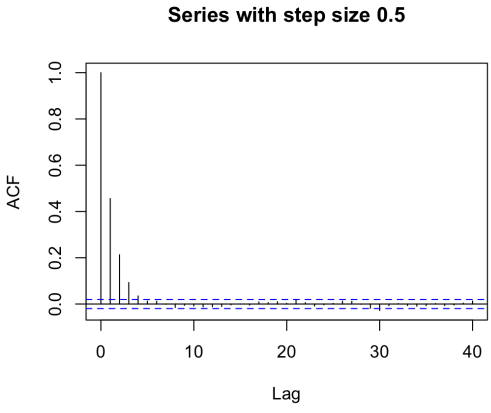


Figure: $q(x, y) = \mathcal{N}\left(\frac{x+0.5}{2}, 0.5^2\right)$