Markov Chain Monte Carlo

Kengo Kamatani

Institute of Statistical Mathematics

1. Ergodicity of Markov chains

1.1 Markov Chains

1.1 Markov Chains

> A Markov chain is a sequence of random variables $X_0, X_1, \ldots$, which extends the concept of independent and identically distributed variables.

1.1 Markov Chains

> A Markov chain is a sequence of random variables $X_0, X_1, \ldots$, which extends the concept of independent and identically distributed variables.

A sequence of random variables is called a **Markov chain** if there exists a function $P(x, \cdot)$ such that, for each $m = 0, 1, 2, \ldots$, the following holds:

$$\mathbb{P}(X_{m+1} \in A \mid X_m = x) = P(x, A)$$

Here, $P(x, \cdot)$ is called the **Markov kernel**.

$P(x, \cdot)$ represents a probability distribution for each x.

$P(x, \cdot)$ represents a probability distribution for each x.

The probability that the **next state** will be in set A, given the current state x, is expressed as:

$$P(x, A) = \mathbb{P}(X_{m+1} \in A \mid X_m = x)$$

$P(x, \cdot)$ represents a probability distribution for each x.

The probability that the **next state** will be in set A, given the current state x, is expressed as:

$$P(x, A) = \mathbb{P}(X_{m+1} \in A \mid X_m = x)$$

Similarly, the probability that the **next-next state** will be in set A, given the current state x, is:

$$P^2(x, A) = \mathbb{P}(X_{m+2} \in A \mid X_m = x)$$

$P(x, \cdot)$ represents a probability distribution for each x.

The probability that the **next state** will be in set A, given the current state x, is expressed as:

$$P(x, A) = \mathbb{P}(X_{m+1} \in A \mid X_m = x)$$

Similarly, the probability that the **next-next state** will be in set A, given the current state x, is:

$$P^2(x, A) = \mathbb{P}(X_{m+2} \in A \mid X_m = x)$$

More generally, for any n,

$$P^n(x, A) = \mathbb{P}(X_{m+n} \in A \mid X_m = x)$$

Two Types of Markov Chains

Two Types of Markov Chains

Now, let's introduce two types of Markov chains:

Two Types of Markov Chains

Now, let's introduce two types of Markov chains:

- In the first type, the behavior (distribution) of X_m remains dependent on the initial value $X_0 = x$, even as m becomes large.

Two Types of Markov Chains

Now, let's introduce two types of Markov chains:

- In the first type, the behavior (distribution) of X_m remains dependent on the initial value $X_0 = x$, even as m becomes large.
- In the second type, this dependency diminishes over time.

Two Types of Markov Chains

Now, let's introduce two types of Markov chains:

- In the first type, the behavior (distribution) of X_m remains dependent on the initial value $X_0 = x$, even as m becomes large.
- In the second type, this dependency diminishes over time.

The first type is called **non-ergodic**, while the second is **ergodic**.

Two Types of Markov Chains

Now, let's introduce two types of Markov chains:

- In the first type, the behavior (distribution) of X_m remains dependent on the initial value $X_0 = x$, even as m becomes large.
- In the second type, this dependency diminishes over time.

The first type is called **non-ergodic**, while the second is **ergodic**.

Let’s focus on initial value dependency.

1.2 Wright-Fisher Model

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

The total number of stones in the jar is N.

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

The total number of stones in the jar is N.

You want to create a new jar by drawing stones from the original jar and making **copies** of them in the new one.

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

The total number of stones in the jar is N.

You want to create a new jar by drawing stones from the original jar and making **copies** of them in the new one.

After you copy a stone, you **put it back** into the original jar, so nothing changes there.

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

The total number of stones in the jar is N.

You want to create a new jar by drawing stones from the original jar and making **copies** of them in the new one.

After you copy a stone, you **put it back** into the original jar, so nothing changes there.

You do this N times, copying one stone each time.

1.2 Wright-Fisher Model

Setup

Imagine you have a jar with two types of stones: **red** and **blue**.

The total number of stones in the jar is N.

You want to create a new jar by drawing stones from the original jar and making **copies** of them in the new one.

After you copy a stone, you **put it back** into the original jar, so nothing changes there.

You do this N times, copying one stone each time.

Once the new jar is filled, the original jar is thrown away, and the process starts again.

Example

If $N = 10$ (there are 10 stones total in the jar), and 3 of those stones are **red**, the chance of picking a **red** stone on any given draw is $3/10 = 30\%$.

Example

If $N = 10$ (there are 10 stones total in the jar), and 3 of those stones are **red**, the chance of picking a **red** stone on any given draw is $3/10 = 30\%$.

The number of **red** stones in the new jar after drawing N times is random but follows a **binomial distribution**, which just means the outcome is based on repeated trials of picking **red** stones with a certain probability. In this case, the number of **red** stones in the new jar follows $\mathcal{B}(N, 0.3)$:

$$\mathbb{P}(X = y) = \binom{10}{y}(0.3)^y(0.7)^{10-y}$$

Example

If $N = 10$ (there are 10 stones total in the jar), and 3 of those stones are **red**, the chance of picking a **red** stone on any given draw is $3/10 = 30\%$.

The number of **red** stones in the new jar after drawing N times is random but follows a **binomial distribution**, which just means the outcome is based on repeated trials of picking **red** stones with a certain probability. In this case, the number of **red** stones in the new jar follows $\mathcal{B}(N, 0.3)$:

$$\mathbb{P}(X = y) = \binom{10}{y}(0.3)^y(0.7)^{10-y}$$

This gives the probability that the new jar will have exactly y **red** stones, where $\binom{N}{y}$ is the binomial coefficient, a way of counting how many ways you can choose y **red** stones out of N draws.

The Process

Now, let's describe this in general terms. Suppose we start with x **red** stones in the jar (so there are $N - x$ **blue** stones).

The Process

Now, let's describe this in general terms. Suppose we start with x **red** stones in the jar (so there are $N - x$ **blue** stones).

In the first trial (creating the first new jar), the number of **red** stones in the new jar is random and follows N indenepdent draw from a binomial distribution $\mathcal{B}(N, x/N)$.

The Process

Now, let's describe this in general terms. Suppose we start with x **red** stones in the jar (so there are $N - x$ **blue** stones).

In the first trial (creating the first new jar), the number of **red** stones in the new jar is random and follows N indenepdent draw from a binomial distribution $\mathcal{B}(N, x/N)$.

For each following trial, the same process happens: the number of **red** stones in the new jar depends on how many **red** stones were in the previous jar.

The Process

Now, let's describe this in general terms. Suppose we start with x **red** stones in the jar (so there are $N - x$ **blue** stones).

In the first trial (creating the first new jar), the number of **red** stones in the new jar is random and follows N indenepdent draw from a binomial distribution $\mathcal{B}(N, x/N)$.

For each following trial, the same process happens: the number of **red** stones in the new jar depends on how many **red** stones were in the previous jar.

If there were x **red** stones before, the number of **red** stones in the new jar still follows a binomial distribution $\mathcal{B}(N, x/N)$.

The Process

Now, let's describe this in general terms. Suppose we start with x **red** stones in the jar (so there are $N - x$ **blue** stones).

In the first trial (creating the first new jar), the number of **red** stones in the new jar is random and follows N indenepdent draw from a binomial distribution $\mathcal{B}(N, x/N)$.

For each following trial, the same process happens: the number of **red** stones in the new jar depends on how many **red** stones were in the previous jar.

If there were x **red** stones before, the number of **red** stones in the new jar still follows a binomial distribution $\mathcal{B}(N, x/N)$.

This process, where the number of **red** stones in each new jar depends on the number from the previous jar, is called the **Wright-Fisher model**.

Summary

- You start with a jar containing **red** and **blue** stones.

Summary

- You start with a jar containing **red** and **blue** stones.
- You create a new jar by randomly copying stones from the original one, and the number of **red** stones in the new jar is based on a binomial distribution.

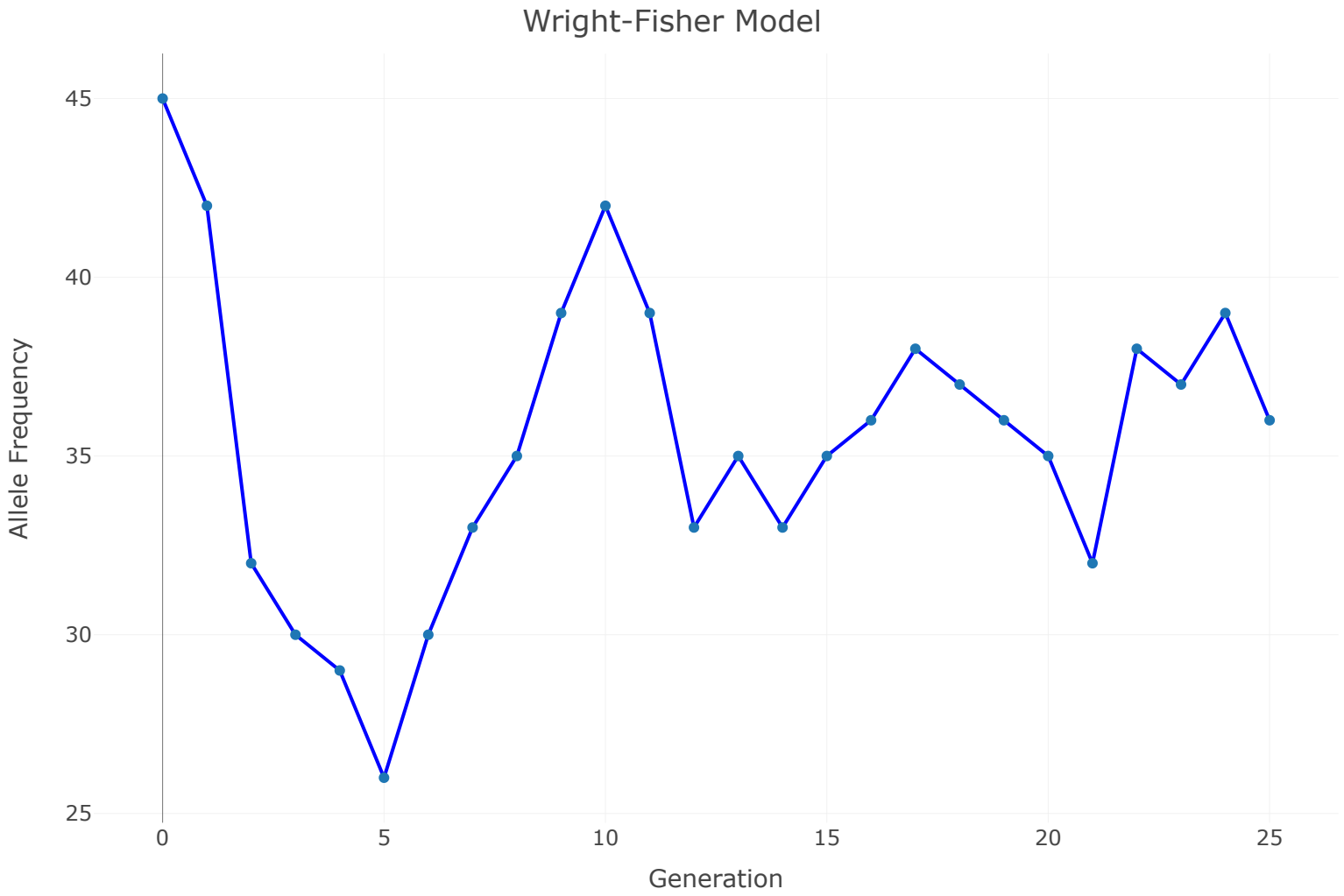
Summary

- You start with a jar containing **red** and **blue** stones.
- You create a new jar by randomly copying stones from the original one, and the number of **red** stones in the new jar is based on a binomial distribution.
- After each trial, the new jar becomes the starting point for the next trial.

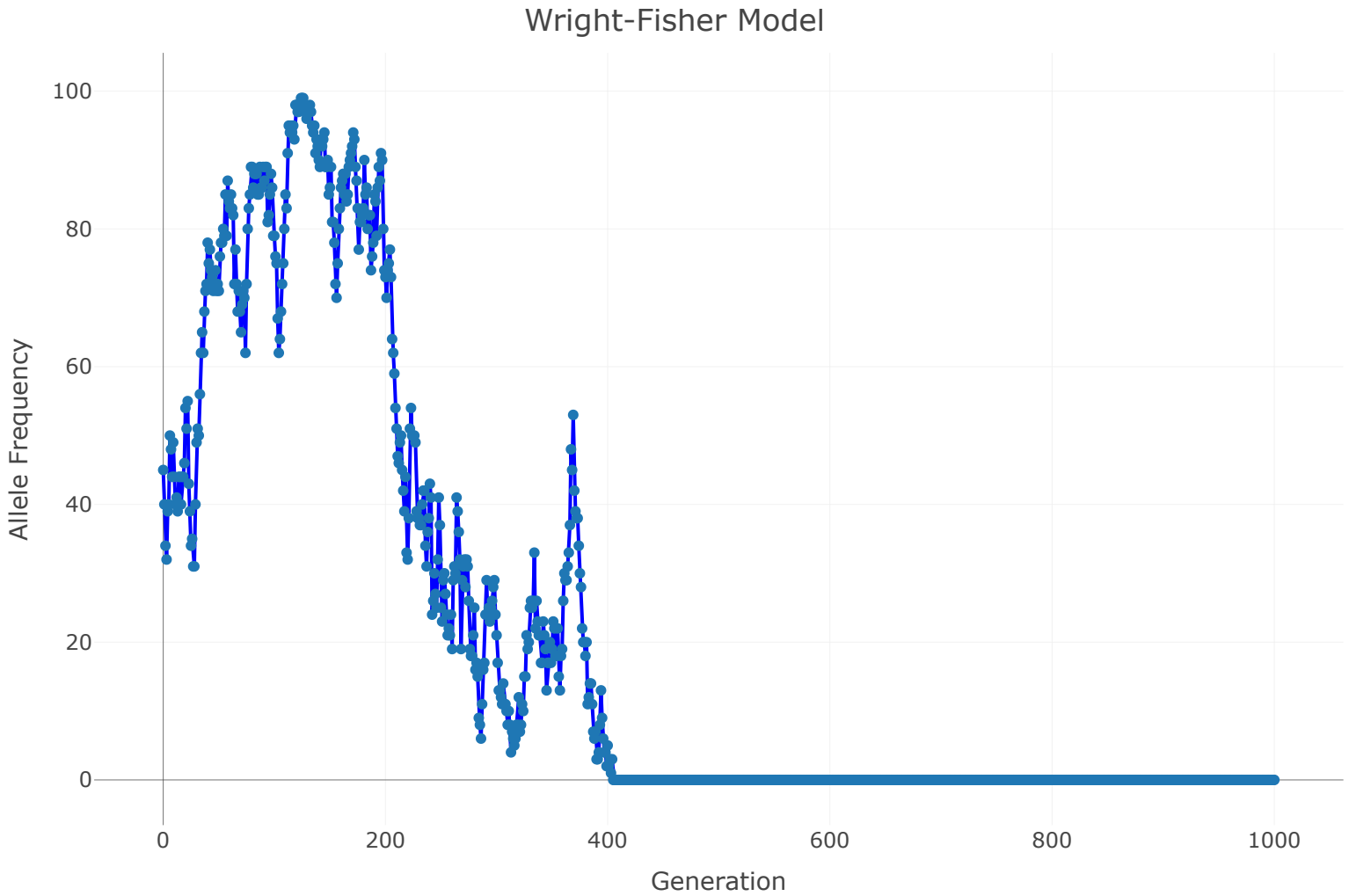
Summary

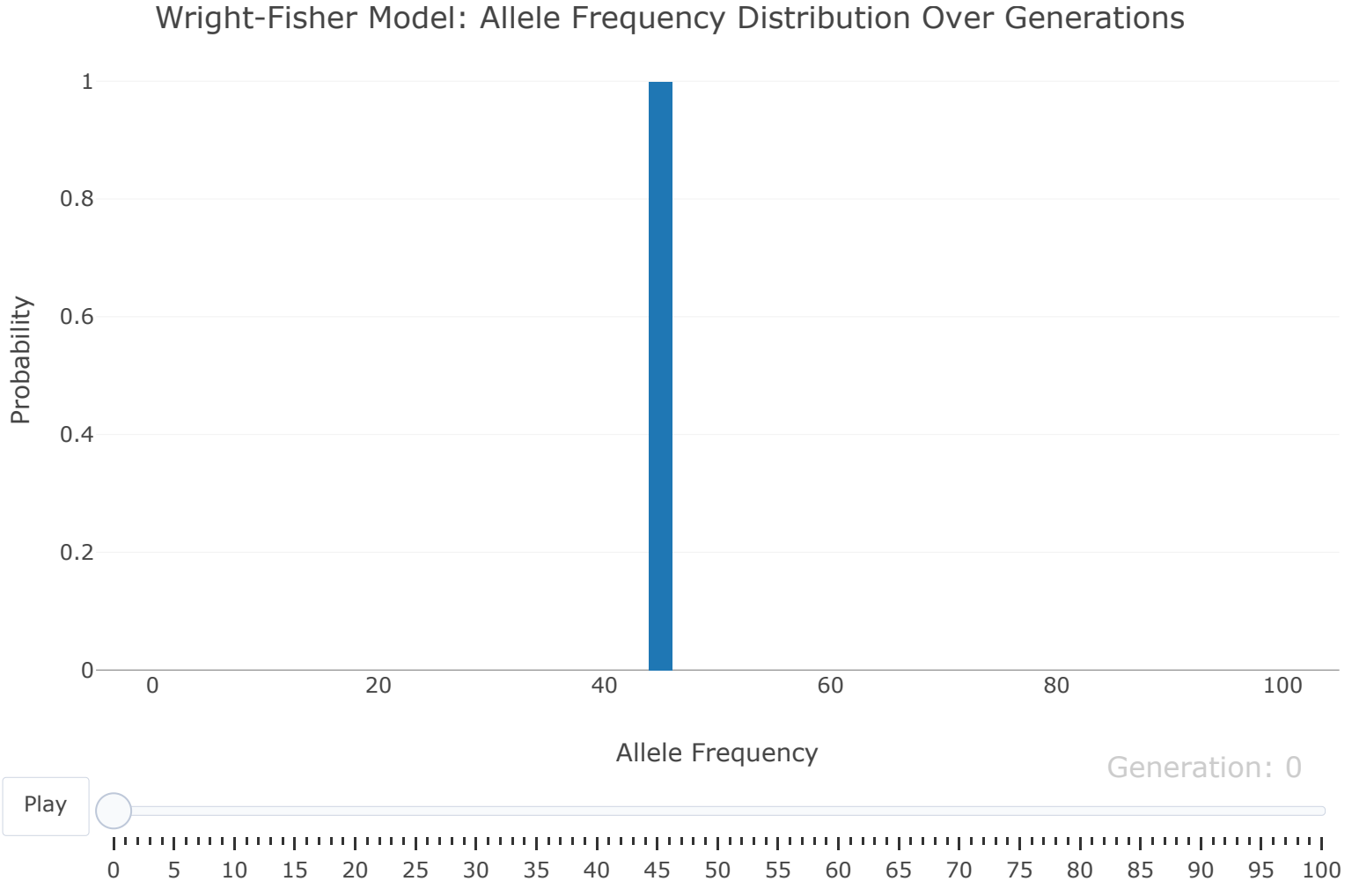
- You start with a jar containing **red** and **blue** stones.
- You create a new jar by randomly copying stones from the original one, and the number of **red** stones in the new jar is based on a binomial distribution.
- After each trial, the new jar becomes the starting point for the next trial.
- The number of **red** stones in each new jar depends on how many **red** stones were in the previous jar.

Let's experiment with this model for $N = 100$, starting with 45 red stones, over 25 trials:



Now, let's extend the experiment to 1000 trials. Note that the number of red stones becomes 0 at some point, and once this happens, the number does not change:




Wright-Fisher Model: Allele Frequency Distribution Over Generations
Probability
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
Allele Frequency
Generation: 0
Play
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

When the number of red stones reaches either 0 or N, it remains unchanged thereafter.

When the number of red stones reaches either 0 or N, it remains unchanged thereafter.

A set like $\{0, N\}$, from which the process cannot escape once entered, is called an **absorbing set**.

When the number of red stones reaches either 0 or N, it remains unchanged thereafter.

A set like $\{0, N\}$, from which the process cannot escape once entered, is called an **absorbing set**.

In the Wright-Fisher model, the final state will end up in an absorbing set, either 0 or N, and the proportion of each is determined probabilistically based on the initial number of red stones.

Exercises 1.

Exercises 1.

1.1 Evaluate $\mathbb{E}[X_{m+1} \mid X_m = x]$ and $\mathbf{Var}(X_{m+1} \mid X_m = x)$

Exercises 1.

1.1 Evaluate $\mathbb{E}[X_{m+1} \mid X_m = x]$ and $\mathbf{Var}(X_{m+1} \mid X_m = x)$

1.2 Let N be a positive integer, and let x be an integer such that $0 \leq x \leq N$. For the Wright-Fisher model, show that:

$$\mathbb{E}[X_{m+1}(N - X_{m+1}) \mid X_m = x] = \left(1 - \frac{1}{N}\right) x(N - x).$$

Exercises 1.

1.1 Evaluate $\mathbb{E}[X_{m+1} \mid X_m = x]$ and $\mathbf{Var}(X_{m+1} \mid X_m = x)$

1.2 Let N be a positive integer, and let x be an integer such that $0 \leq x \leq N$. For the Wright-Fisher model, show that:

$$\mathbb{E}[X_{m+1}(N - X_{m+1}) \mid X_m = x] = \left(1 - \frac{1}{N}\right) x(N - x).$$

1.3 From the above equation, conclude that X_m converges in probability to 0 or N when $m \to \infty$.

1.3 Autoregressive Model

1.3 Autoregressive Model

A Markov chain defined by

$$X_{m+1} = \alpha X_m + W_{m+1}, \quad W_{m+1} \sim \mathcal{N}(0, \sigma^2),$$

where $W_1, W_2, \ldots$ are independent, is called an **autoregressive model**.

1.3 Autoregressive Model

A Markov chain defined by

$$X_{m+1} = \alpha X_m + W_{m+1}, \quad W_{m+1} \sim \mathcal{N}(0, \sigma^2),$$

where $W_1, W_2, \ldots$ are independent, is called an **autoregressive model**.

From this equation, we can derive:

$$X_m = \alpha^m X_0 + \sum_{n=0}^{m-1} \alpha^n W_{m-n}.$$

Due to the reproductive property of normal distributions, we find that X_m follows a normal distribution:

$$X_m \sim \mathcal{N}\left(\alpha^m X_0, \frac{1-\alpha^{2m}}{1-\alpha^2}\sigma^2\right),$$

Due to the reproductive property of normal distributions, we find that X_m follows a normal distribution:

$$X_m \sim \mathcal{N}\left(\alpha^m X_0, \frac{1-\alpha^{2m}}{1-\alpha^2}\sigma^2\right),$$

and if $|\alpha| < 1$,

$$X_m \to \mathcal{N}(0, \frac{\sigma^2}{1-\alpha^2}) \quad \text{as} \quad m \to \infty.$$

Due to the reproductive property of normal distributions, we find that X_m follows a normal distribution:

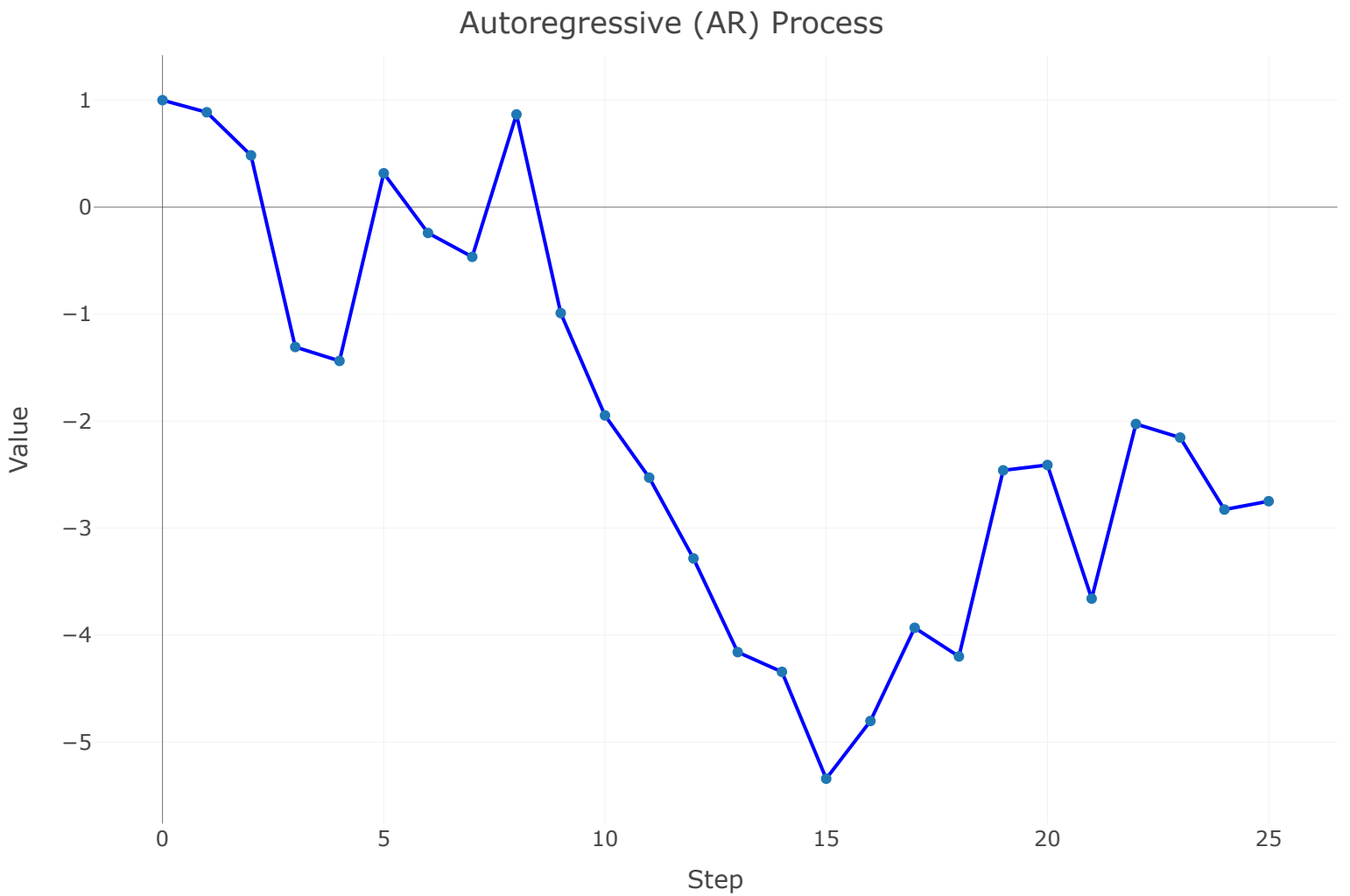
$$X_m \sim \mathcal{N}\left(\alpha^m X_0, \frac{1-\alpha^{2m}}{1-\alpha^2}\sigma^2\right),$$

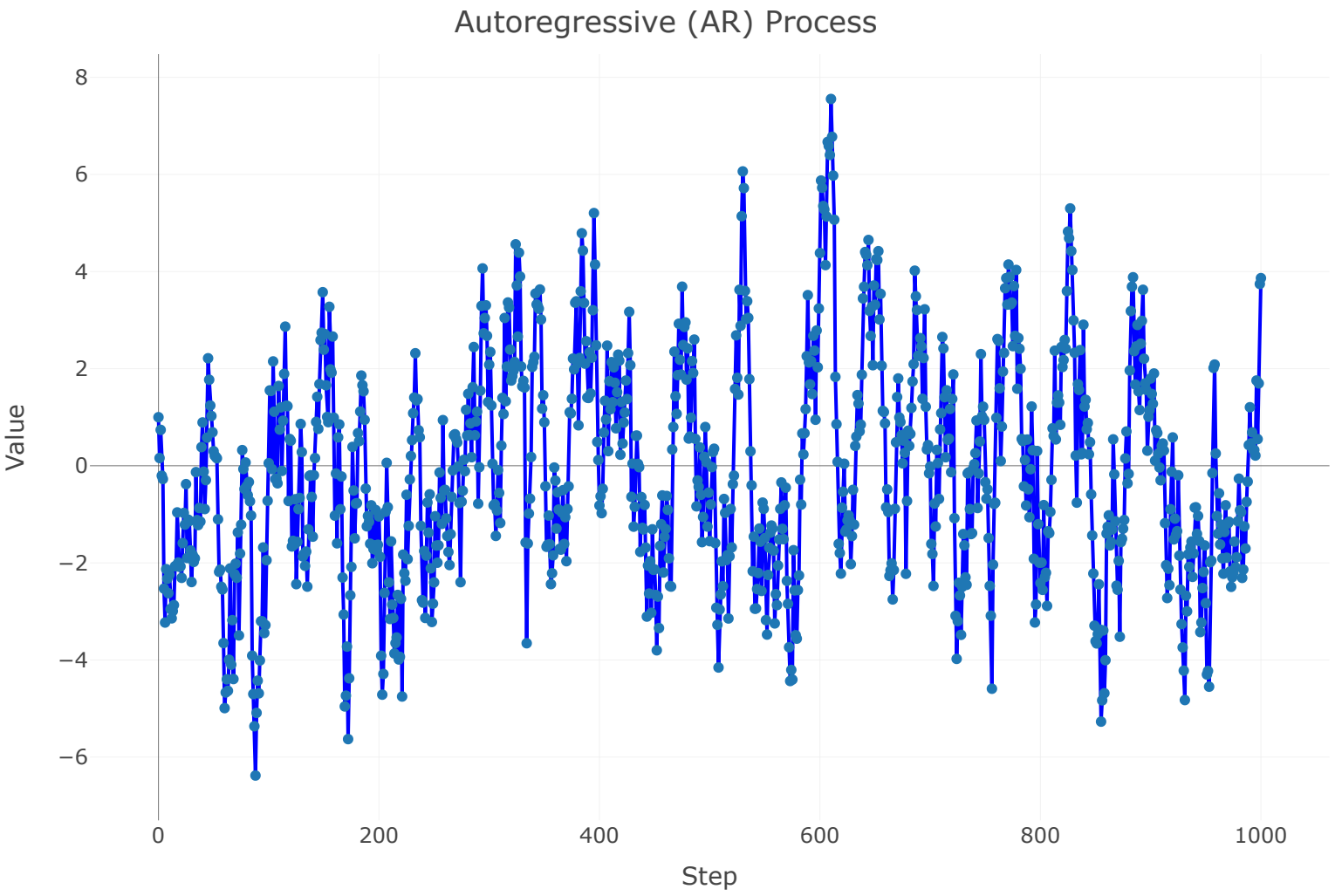
and if $|\alpha| < 1$,

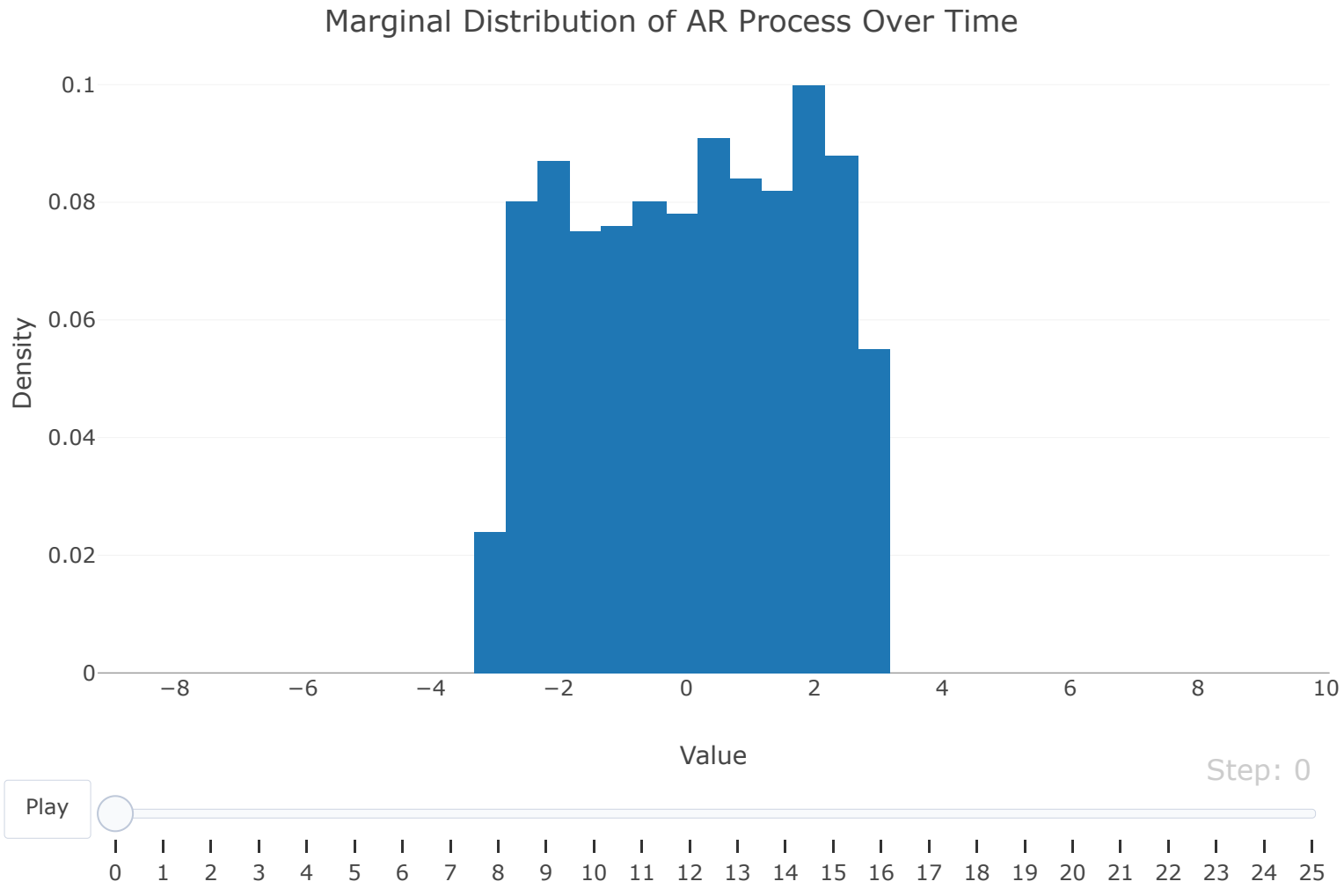
$$X_m \to \mathcal{N}(0, \frac{\sigma^2}{1-\alpha^2}) \quad \text{as} \quad m \to \infty.$$

The autoregressive model differs from the Wright-Fisher model in that the limiting distribution does not depend on the initial value.

Example: Autoregressive Process




Autoregressive (AR) Process
Value
Step
8
6
4
2
0
−2
−4
−6
0
200
400
600
800
1000


Marginal Distribution of AR Process Over Time
Density
0.1
0.08
0.06
0.04
0.02
0
−8
−6
−4
−2
0
2
4
6
8
10
Value
Play
Step: 0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Summary

Summary

- We introduced two types of Markov chains: the Wright-Fisher model and the Autoregressive model.

Summary

- We introduced two types of Markov chains: the Wright-Fisher model and the Autoregressive model.
- These two chains exhibit distinctly different behaviors.

Summary

- We introduced two types of Markov chains: the Wright-Fisher model and the Autoregressive model.
- These two chains exhibit distinctly different behaviors.
- In the Wright-Fisher model, the marginal distribution remains influenced by the initial state, even after a large number of iterations.

Summary

- We introduced two types of Markov chains: the Wright-Fisher model and the Autoregressive model.
- These two chains exhibit distinctly different behaviors.
- In the Wright-Fisher model, the marginal distribution remains influenced by the initial state, even after a large number of iterations.
- In contrast, the Autoregressive model shows diminishing dependence on the initial state as the number of iterations increases, with the system gradually stabilizing.

Summary

- We introduced two types of Markov chains: the Wright-Fisher model and the Autoregressive model.
- These two chains exhibit distinctly different behaviors.
- In the Wright-Fisher model, the marginal distribution remains influenced by the initial state, even after a large number of iterations.
- In contrast, the Autoregressive model shows diminishing dependence on the initial state as the number of iterations increases, with the system gradually stabilizing.

Let’s continue with a more detailed explanation of ergodicity and the concepts involved in the behavior of Markov chains.

1.4 Ergodicity Explained

1.4 Ergodicity Explained

Now that we have introduced two examples:

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).
- One where the influence diminishes (**ergodic** behavior).

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).
- One where the influence diminishes (**ergodic** behavior).

To explain these differences in more details, we need to introduce three important terms:

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).
- One where the influence diminishes (**ergodic** behavior).

To explain these differences in more details, we need to introduce three important terms:

1. **Invariant Probability Measure**

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).
- One where the influence diminishes (**ergodic** behavior).

To explain these differences in more details, we need to introduce three important terms:

1. **Invariant Probability Measure**
2. **Singularity**

1.4 Ergodicity Explained

Now that we have introduced two examples:

- One where the influence of the initial value persists (**non-ergodic** behavior).
- One where the influence diminishes (**ergodic** behavior).

To explain these differences in more details, we need to introduce three important terms:

1. **Invariant Probability Measure**
2. **Singularity**
3. **Total Variation**

Invariant Probability Measure

Invariant Probability Measure

> An **invariant probability measure** is a measure that remains unchanged under the transition induced by a Markov kernel. It serves as the limiting measure of an ergodic Markov chain. However, the converse does not hold true.

Invariant Probability Measure

> An **invariant probability measure** is a measure that remains unchanged under the transition induced by a Markov kernel. It serves as the limiting measure of an ergodic Markov chain. However, the converse does not hold true.

If a Markov chain is ergodic, as in the case of the autoregressive model where $|\alpha| < 1$, there exists a limiting distribution for X_m, denoted by Π.

Invariant Probability Measure

> An **invariant probability measure** is a measure that remains unchanged under the transition induced by a Markov kernel. It serves as the limiting measure of an ergodic Markov chain. However, the converse does not hold true.

If a Markov chain is ergodic, as in the case of the autoregressive model where $|\alpha| < 1$, there exists a limiting distribution for X_m, denoted by Π.

If $X_0 \sim \Pi$, then $X_1 \sim \Pi$ must also hold. A probability distribution Π that satisfies this property is called an **invariant probability measure**.

Singularity

Singularity

> Probability measures are said to be **mutually singular** if they are entirely distinct. We will assume that the Markov chains starting from different points are **not** mutually singular, ensuring that the chains mix well.

Singularity

> Probability measures are said to be **mutually singular** if they are entirely distinct. We will assume that the Markov chains starting from different points are **not** mutually singular, ensuring that the chains mix well.

Two probability distributions P and Q are said to be **mutually singular** if there exists a set A such that:

$$P(A) = 0 \quad \text{and} \quad Q(A^c) = 0.$$

Singularity

> Probability measures are said to be **mutually singular** if they are entirely distinct. We will assume that the Markov chains starting from different points are **not** mutually singular, ensuring that the chains mix well.

Two probability distributions P and Q are said to be **mutually singular** if there exists a set A such that:

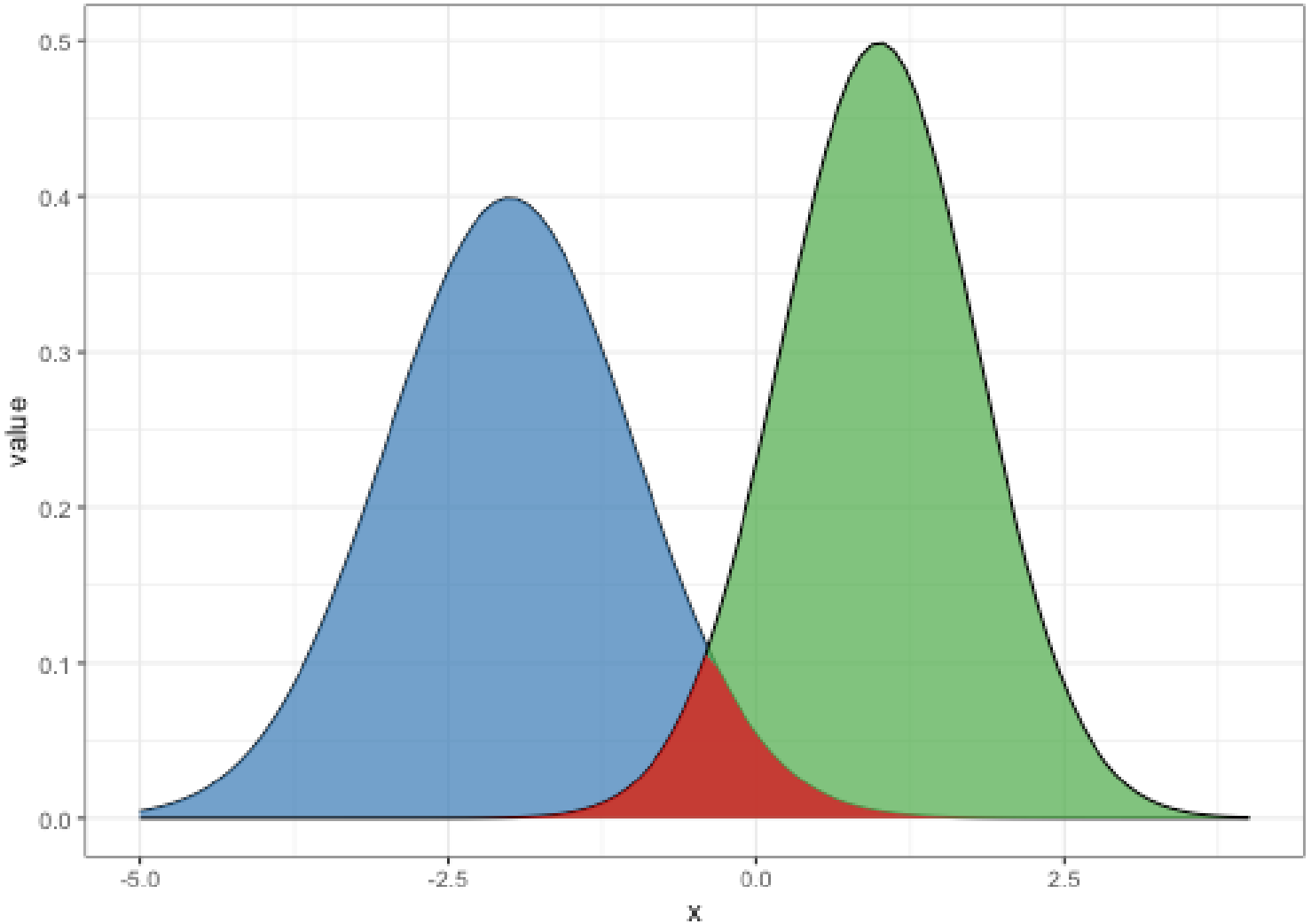
$$P(A) = 0 \quad \text{and} \quad Q(A^c) = 0.$$

In this case, we write $P \perp Q$. If P and Q are not mutually singular, we write $P \not\perp Q$.

Let’s visualize this property on the next page using the overlapping areas of two probability density functions.

Let's visualize this property on the next page using the overlapping areas of two probability density functions.

If there is a shared **red region** under both probability density functions, then $P \not\perp Q$. If no such region exists, then $P \perp Q$.


0.5
0.4
0.3
0.2
0.1
0.0
value
-5.0
-2.5
0.0
2.5
x

Total Variation

Total Variation

> Total variation distance measures the difference between two probability distributions. It can be visualized as the area between the two probability density functions.

Total Variation

> Total variation distance measures the difference between two probability distributions. It can be visualized as the area between the two probability density functions.

We define the total variation distance between two probability distributions P and Q as:

$$\|P - Q\|_{\mathrm{TV}} = \int_E |p(x) - q(x)| \, \mathrm{d}x.$$

Total Variation

> Total variation distance measures the difference between two probability distributions. It can be visualized as the area between the two probability density functions.

We define the total variation distance between two probability distributions P and Q as:

$$\|P - Q\|_{\mathrm{TV}} = \int_E |p(x) - q(x)| \, \mathrm{d}x.$$

This measures the difference between the probability density functions $p(x)$ and $q(x)$.

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

$$\|P - Q\|_{\mathrm{TV}} = \text{Blue Region} + \text{Green Region}.$$

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

$$\|P - Q\|_{\mathrm{TV}} = \text{Blue Region} + \text{Green Region}.$$

Since the total area under each probability density function is 1, we have:

$$1 = \text{Blue Region} + \text{Red Region},$$

and similarly for the green region:

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

$$\|P - Q\|_{\mathrm{TV}} = \text{Blue Region} + \text{Green Region}.$$

Since the total area under each probability density function is 1, we have:

$$1 = \text{Blue Region} + \text{Red Region},$$

and similarly for the green region:

$$1 = \text{Green Region} + \text{Red Region}.$$

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

$$\|P - Q\|_{\text{TV}} = \text{Blue Region} + \text{Green Region}.$$

Since the total area under each probability density function is 1, we have:

$$1 = \text{Blue Region} + \text{Red Region},$$

and similarly for the green region:

$$1 = \text{Green Region} + \text{Red Region}.$$

Thus, the areas of the blue and green regions are equal:

$$\text{Blue Region} = \text{Green Region} = \frac{1}{2}\|P - Q\|_{\text{TV}}.$$

The total variation distance is the sum of the **blue region** and **green region** in the diagram, representing the areas under each function.

$$\|P - Q\|_{\mathrm{TV}} = \text{Blue Region} + \text{Green Region}.$$

Since the total area under each probability density function is 1, we have:

$$1 = \text{Blue Region} + \text{Red Region},$$

and similarly for the green region:

$$1 = \text{Green Region} + \text{Red Region}.$$

Thus, the areas of the blue and green regions are equal:

$$\text{Blue Region} = \text{Green Region} = \frac{1}{2}\|P - Q\|_{\mathrm{TV}}.$$

Hence, the area of the red region is:

$$\text{Red Region} = 1 - \frac{1}{2}\|P - Q\|_{\mathrm{TV}}.$$

If $P \perp Q$, the area of the red region is 0, meaning that $\|P - Q\|_{\mathrm{TV}} = 2$, and conversely, if $P \not\perp Q$, then $\|P - Q\|_{\mathrm{TV}} < 2$.

Summary

Summary

- An **invariant probability measure** is a measure that remains unchanged under the transition dynamics of a Markov chain.

Summary

- An **invariant probability measure** is a measure that remains unchanged under the transition dynamics of a Markov chain.
- If the red region of the graph is zero, we say that the two probability measures are **mutually singular**.

Summary

- An **invariant probability measure** is a measure that remains unchanged under the transition dynamics of a Markov chain.
- If the red region of the graph is zero, we say that the two probability measures are **mutually singular**.
- This red region can be expressed as $1 - \|P - Q\|_{\mathrm{TV}}/2$, where $\|P - Q\|_{\mathrm{TV}}$ is the total variation distance between the measures P and Q.

Exercise 2.

2.1 For a real number μ, find the total variation distance $\|\mathcal{N}(0,1) - \mathcal{N}(\mu,1)\|_{\mathrm{TV}}$.

Exercise 2.

2.1 For a real number μ, find the total variation distance $\|\mathcal{N}(0,1) - \mathcal{N}(\mu,1)\|_{\mathrm{TV}}$.

2.2 For a real number $\sigma > 0$, find the total variation distance $\|\mathcal{N}(0,1) - \mathcal{N}(0,\sigma^2)\|_{\mathrm{TV}}$.

Exercise 2.

2.1 For a real number μ, find the total variation distance $\|\mathcal{N}(0,1) - \mathcal{N}(\mu,1)\|_{\mathrm{TV}}$.

2.2 For a real number $\sigma > 0$, find the total variation distance $\|\mathcal{N}(0,1) - \mathcal{N}(0,\sigma^2)\|_{\mathrm{TV}}$.

2.3 Prove that no invariant probability measure exists for the Autoregressive model when $\alpha = 1$. (Hint: Evaluate the characteristic function. Recall that every characteristic function $\psi(u)$ satisfies $\psi(0) = 1$ and is continuous at $u = 0$.)

1.5 Theorem (The law of large numbers):

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

Further, suppose there exists an **invariant probability distribution** Π.

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

Further, suppose there exists an **invariant probability distribution** Π.

Then, for almost all points x and sets A:

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

Further, suppose there exists an **invariant probability distribution** Π.

Then, for almost all points x and sets A:

$$P^m(x, A) \longrightarrow_{m \to \infty} \Pi(A).$$

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

Further, suppose there exists an **invariant probability distribution** Π.

Then, for almost all points x and sets A:

$$P^m(x, A) \longrightarrow_{m \to \infty} \Pi(A).$$

Moreover, if $X_0 = x$, the law of large numbers holds:

1.5 Theorem (The law of large numbers):

For any points x and y, assume $P(x, \cdot) \not\perp P(y, \cdot)$.

Further, suppose there exists an **invariant probability distribution** Π.

Then, for almost all points x and sets A:

$$P^m(x, A) \longrightarrow_{m \to \infty} \Pi(A).$$

Moreover, if $X_0 = x$, the law of large numbers holds:

$$\frac{1}{M} \sum_{m=0}^{M-1} f(X_m) \longrightarrow_{M \to \infty} \int f(x) \Pi(\mathrm{d}x),$$

as long as the right-hand side exists.

Proof Outline:

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2}\|P(x, \cdot) - P(y, \cdot)\|_{\mathrm{TV}}.$$

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2}\|P(x, \cdot) - P(y, \cdot)\|_{\mathrm{TV}}.$$

It turns out that, for $n = 1, 2, \ldots$, the area of the red region formed by $P^n(x, \cdot)$ and $P^n(y, \cdot)$ **increases** monotonically with respect to n.

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2}\|P(x, \cdot) - P(y, \cdot)\|_{\mathrm{TV}}.$$

It turns out that, for $n = 1, 2, \ldots$, the area of the red region formed by $P^n(x, \cdot)$ and $P^n(y, \cdot)$ **increases** monotonically with respect to n.

The proof concludes when the red region occupies 100% of the area, meaning that $P^n(x, \cdot)$ and $P^n(y, \cdot)$ coincide completely as $n \to \infty$.

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2}\|P(x, \cdot) - P(y, \cdot)\|_{\mathrm{TV}}.$$

It turns out that, for $n = 1, 2, \ldots$, the area of the red region formed by $P^n(x, \cdot)$ and $P^n(y, \cdot)$ **increases** monotonically with respect to n.

The proof concludes when the red region occupies 100% of the area, meaning that $P^n(x, \cdot)$ and $P^n(y, \cdot)$ coincide completely as $n \to \infty$.

It is easy to see that this limiting distribution matches Π.

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2}\|P(x, \cdot) - P(y, \cdot)\|_{\mathrm{TV}}.$$

It turns out that, for $n = 1, 2, \ldots$, the area of the red region formed by $P^n(x, \cdot)$ and $P^n(y, \cdot)$ **increases** monotonically with respect to n.

The proof concludes when the red region occupies 100% of the area, meaning that $P^n(x, \cdot)$ and $P^n(y, \cdot)$ coincide completely as $n \to \infty$.

It is easy to see that this limiting distribution matches Π.

> Technically, it is **not easy** to directly observe the total variation distance. The **coupling technique** is a useful method for estimating the quantity, and we will briefly discuss it.

Proof Outline:

The proof proceeds by considering the area of the **red region** formed by $P(x, \cdot)$ and $P(y, \cdot)$, which is given by:

$$1 - \frac{1}{2} \| P(x, \cdot) - P(y, \cdot) \|_{\mathrm{TV}}.$$

It turns out that, for $n = 1, 2, \ldots$, the area of the red region formed by $P^n(x, \cdot)$ and $P^n(y, \cdot)$ **increases** monotonically with respect to n.

The proof concludes when the red region occupies 100% of the area, meaning that $P^n(x, \cdot)$ and $P^n(y, \cdot)$ coincide completely as $n \to \infty$.

It is easy to see that this limiting distribution matches Π.

> Technically, it is **not easy** to directly observe the total variation distance. The **coupling technique** is a useful method for estimating the quantity, and we will briefly discuss it.

For more details, see "Kulik (2017)" Theorem 2.5.1.

1.6 Coupling

1.6 Coupling

> The total varation leads us to **coupling**, a technique to measure the closeness of distributions by relating them to the closeness of the corresponding random variables.

1.6 Coupling

> The total varation leads us to **coupling**, a technique to measure the closeness of distributions by relating them to the closeness of the corresponding random variables.

Consider two continuous probability distributions P and Q on the real line.

1.6 Coupling

> The total varation leads us to **coupling**, a technique to measure the closeness of distributions by relating them to the closeness of the corresponding random variables.

Consider two continuous probability distributions P and Q on the real line.

If X and Y are independent and distributed according to P and Q, respectively, then:

$$\mathbb{P}(X = Y) = 0,$$

1.6 Coupling

> The total varation leads us to **coupling**, a technique to measure the closeness of distributions by relating them to the closeness of the corresponding random variables.

Consider two continuous probability distributions P and Q on the real line.

If X and Y are independent and distributed according to P and Q, respectively, then:

$$\mathbb{P}(X = Y) = 0,$$

even if P and Q are very close or even identical.

1.6 Coupling

> The total varation leads us to **coupling**, a technique to measure the closeness of distributions by relating them to the closeness of the corresponding random variables.

Consider two continuous probability distributions P and Q on the real line.

If X and Y are independent and distributed according to P and Q, respectively, then:

$$\mathbb{P}(X = Y) = 0,$$

even if P and Q are very close or even identical.

However, with **coupling**, we can construct the pair (X, Y) in such a way that we increase the probability $\mathbb{P}(X = Y)$ or reduce the difference $|X - Y|$.

Coupling (continued)

Coupling is a method for constructing a pair (X, Y), where $X \sim P$ and $Y \sim Q$, in such a way that we increase the probability $\mathbb{P}(X = Y)$ or make $|X - Y|$ smaller.

In Markov chain convergence theory, **optimal coupling** plays an important role. Optimal coupling maximizes $\mathbb{P}(X = Y)$, which is beneficial for proving convergence.

Imagine we have 100 **German** and 100 Japanese male Judo players (Judoka). Our goal is to pair them by matching those in the same weight class. Ideally, we want to create as many matches where the weight classes of the players, X and Y, are identical, meaning $X = Y$.

The beautiful image was shared by Joshua Jamias on Unsplash

However, even with careful pairing, there’s a limit to the number of same-class matches we can create.

However, even with careful pairing, there's a limit to the number of same-class matches we can create.

For example:

- If there are 3 **German** and 5 **Japanese** players in the under-66 kg weight class, the maximum number of same-class matches is 3.
- If there are 6 **German** and 2 **Japanese** players in the 81 kg weight class, the maximum number of same-class matches is 2.

However, even with careful pairing, there's a limit to the number of same-class matches we can create.

For example:

- If there are 3 **German** and 5 **Japanese** players in the under-66 kg weight class, the maximum number of same-class matches is 3.
- If there are 6 **German** and 2 **Japanese** players in the 81 kg weight class, the maximum number of same-class matches is 2.

Therefore, the total number of same-class matches is determined by the smaller number of players from each group in every weight class. Any remaining players will participate in mixed-class matches, where $X \neq Y$.

The proportion of same-class matches corresponds to the **red region** in the plot, representing the overlap of the probability densities.

The proportion of same-class matches corresponds to the **red region** in the plot, representing the overlap of the probability densities.

Mathematically, when two distributions P and Q are coupled, the maximum probability of forming a match $X = Y$ is:

$$\mathbb{P}(X = Y) = 1 - \frac{1}{2}\|P - Q\|_{\mathrm{TV}}.$$

This coupling, which achieves this maximum probability, is called **optimal coupling**.

Sequential Coupling in Markov Chains

Now, let's apply coupling sequentially to Markov chains. Consider two Markov chains starting from different initial values x_0 and y_0.

Sequential Coupling in Markov Chains

Now, let's apply coupling sequentially to Markov chains. Consider two Markov chains starting from different initial values x_0 and y_0.

From the initial values, we generate the next states (X_1, Y_1) using coupling:

$$P(x_0, \cdot), P(y_0, \cdot).$$

Sequential Coupling in Markov Chains

Now, let's apply coupling sequentially to Markov chains. Consider two Markov chains starting from different initial values x_0 and y_0.

From the initial values, we generate the next states (X_1, Y_1) using coupling:

$$P(x_0, \cdot), P(y_0, \cdot).$$

From these new states $(X_1 = x_1, Y_1 = y_1)$, we generate the next pair (X_2, Y_2) using coupling:

$$P(x_1, \cdot), P(y_1, \cdot).$$

Sequential Coupling in Markov Chains

Now, let's apply coupling sequentially to Markov chains. Consider two Markov chains starting from different initial values x_0 and y_0.

From the initial values, we generate the next states (X_1, Y_1) using coupling:

$$P(x_0, \cdot), P(y_0, \cdot).$$

From these new states $(X_1 = x_1, Y_1 = y_1)$, we generate the next pair (X_2, Y_2) using coupling:

$$P(x_1, \cdot), P(y_1, \cdot).$$

This process is repeated indefinitely.

Sequential Coupling in Markov Chains

Now, let's apply coupling sequentially to Markov chains. Consider two Markov chains starting from different initial values x_0 and y_0.

From the initial values, we generate the next states (X_1, Y_1) using coupling:

$$P(x_0, \cdot), P(y_0, \cdot).$$

From these new states $(X_1 = x_1, Y_1 = y_1)$, we generate the next pair (X_2, Y_2) using coupling:

$$P(x_1, \cdot), P(y_1, \cdot).$$

This process is repeated indefinitely.

Once $X_m = Y_m$, they remain coupled for all future steps.

The inequality

$$\mathbb{P}(X_m = Y_m) \leq 1 - \frac{1}{2}\|P^m(x, \cdot) - P^m(y, \cdot)\|_{\mathrm{TV}}.$$

is always true.

The inequality

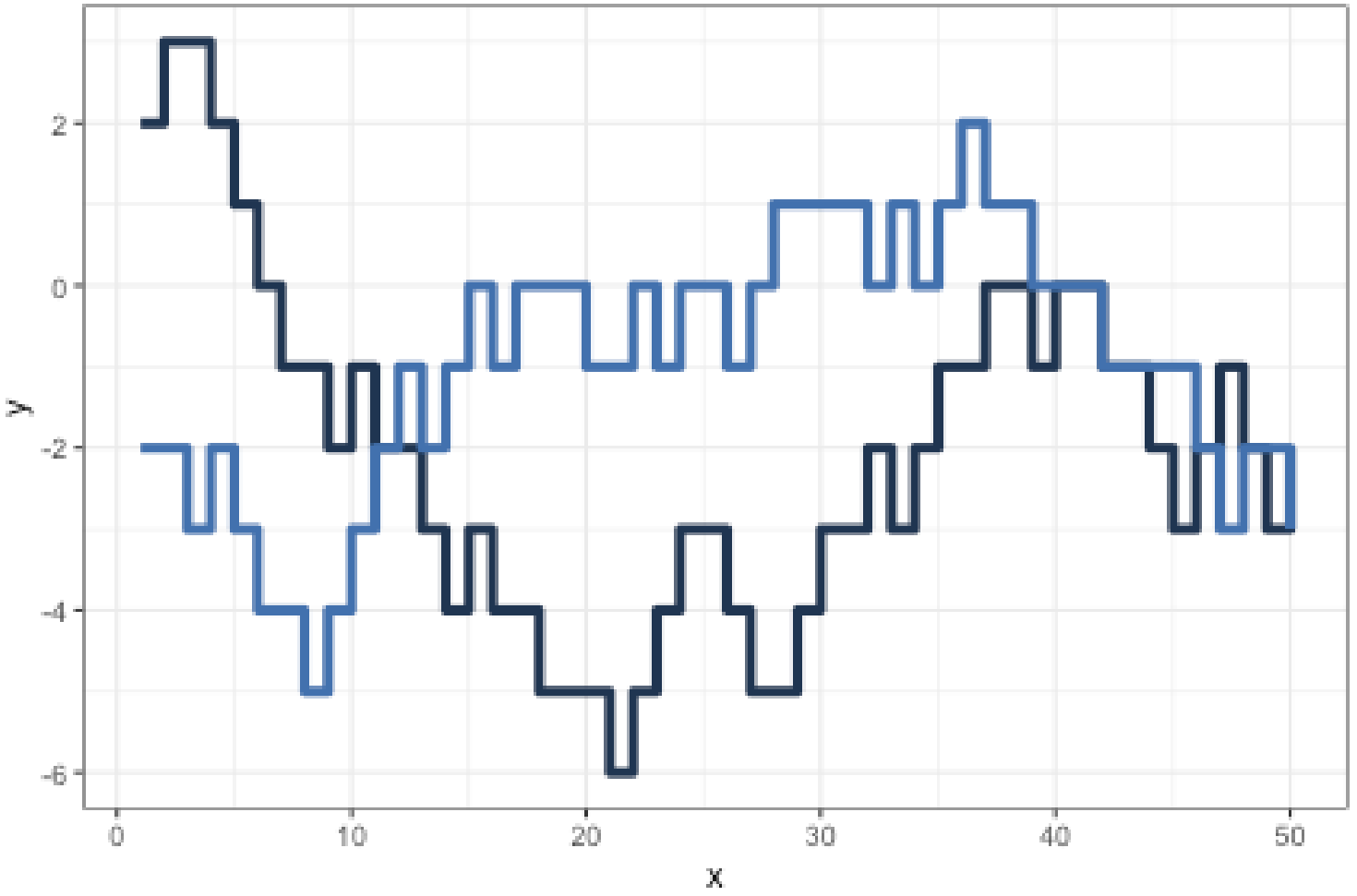
$$\mathbb{P}(X_m = Y_m) \leq 1 - \frac{1}{2}\|P^m(x,\cdot) - P^m(y,\cdot)\|_{\mathrm{TV}}.$$

is always true.

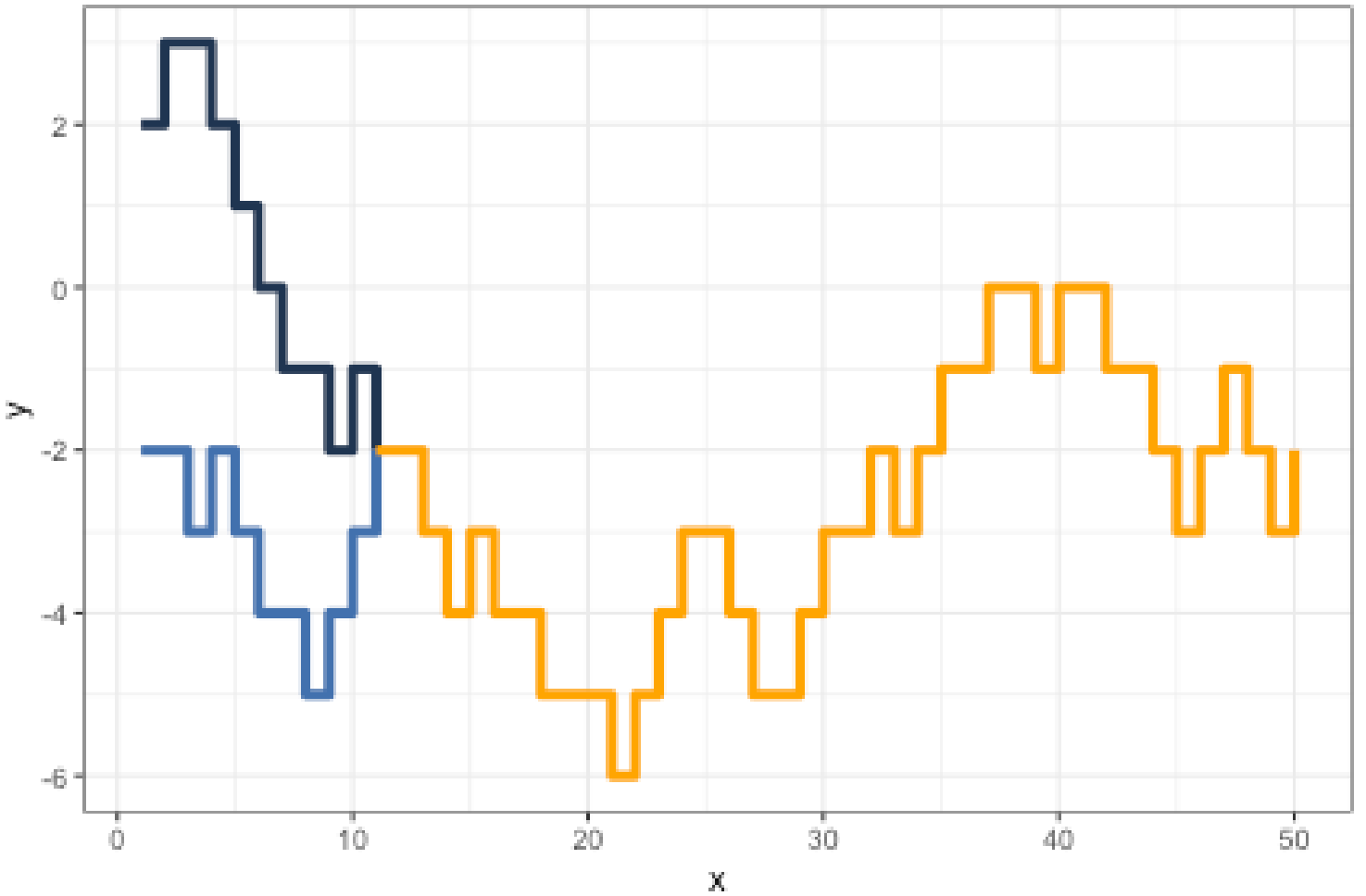
Through coupling, the left-hand side provides a good bound of the right-hand side, and is **easier** to estimate.

Coupling Before and After

Before coupling (left side of the plot):



After coupling (right side of the plot):



Summary

Summary

- The **law of large numbers** applies to Markov chains as well.

Summary

- The **law of large numbers** applies to Markov chains as well.
- A sufficient condition for this is that $P(x, \cdot)$ and $P(y, \cdot)$ are not mutually singular, and that an invariant probability measure exists.

Summary

- The **law of large numbers** applies to Markov chains as well.
- A sufficient condition for this is that $P(x, \cdot)$ and $P(y, \cdot)$ are not mutually singular, and that an invariant probability measure exists.
- **Coupling** is a powerful technique used to establish the law of large numbers in Markov chains.

From here ...

From here ...

- We introduced the law of large numbers for Markov chains.

From here ...

- We introduced the law of large numbers for Markov chains.
- From here, we will introduce a Monte Carlo method using the theorem.

2. Markov chain Monte Carlo

2. Markov chain Monte Carlo

> The **Markov Chain Monte Carlo (MCMC)** method is a powerful numerical technique used to approximate expectations by leveraging the law of large numbers of Markov chains. It generates samples from a target distribution by constructing a Markov chain that converges to this distribution, allowing for the estimation of complex integrals or probabilistic quantities that are otherwise intractable

2.1 Metropolis Algorithm

2.1 Metropolis Algorithm

> The **Metropolis algorithm** is a core component of MCMC, which constructs a **reversible** Markov chain to sample from a target distribution.

2.1 Metropolis Algorithm

> The **Metropolis algorithm** is a core component of MCMC, which constructs a **reversible** Markov chain to sample from a target distribution.

Let the state space E be a discrete set, and let Π be the probability distribution of interest, with probability function $\pi(x)$.

2.1 Metropolis Algorithm

> The **Metropolis algorithm** is a core component of MCMC, which constructs a **reversible** Markov chain to sample from a target distribution.

Let the state space E be a discrete set, and let Π be the probability distribution of interest, with probability function $\pi(x)$.

We attempt to transition from x to y with transition probability:

$$q(x, y).$$

2.1 Metropolis Algorithm

> The **Metropolis algorithm** is a core component of MCMC, which constructs a **reversible** Markov chain to sample from a target distribution.

Let the state space E be a discrete set, and let Π be the probability distribution of interest, with probability function $\pi(x)$.

We attempt to transition from x to y with transition probability:

$$q(x, y).$$

Now, assume stationarity has been reached, meaning that the random variable X satisfies $\mathbb{P}(X = x) = \pi(x)$.

The **outflow** from x to y in one step is the original probability $\pi(x)$ multiplied by the transition probability $q(x, y)$:

$$\pi(x)q(x, y).$$

The **outflow** from x to y in one step is the original probability $\pi(x)$ multiplied by the transition probability $q(x, y)$:

$$\pi(x)q(x, y).$$

Conversely, the **inflow** from y to x is:

$$\pi(y)q(y, x).$$

Introducing Traffic Control

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

$$\pi(x)q(x,y) < \pi(y)q(y,x),$$

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

$$\pi(x)q(x,y) < \pi(y)q(y,x),$$

then we don't prevent the transition from x to y.

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

$$\pi(x)q(x,y) < \pi(y)q(y,x),$$

then we don't prevent the transition from x to y.

On the other hand, if the **outflow** from x to y is greater, we control the **inflow** and **outflow** by allowing the transition with a probability of:

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

$$\pi(x)q(x,y) < \pi(y)q(y,x),$$

then we don't prevent the transition from x to y.

On the other hand, if the **outflow** from x to y is greater, we control the **inflow** and **outflow** by allowing the transition with a probability of:

$$\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}.$$

Introducing Traffic Control

Let's introduce some **traffic control** measures. If the **outflow** from x to y is less than the **inflow** from y to x, that is,

$$\pi(x)q(x,y) < \pi(y)q(y,x),$$

then we don't prevent the transition from x to y.

On the other hand, if the **outflow** from x to y is greater, we control the **inflow** and **outflow** by allowing the transition with a probability of:

$$\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}.$$

If the transition is not allowed, the process stays at x.

To summarize the two cases:

To summarize the two cases:

- If $\pi(x)q(x,y) \leq \pi(y)q(y,x)$, we allow the transition from x to y without restriction.

To summarize the two cases:

- If $\pi(x)q(x,y) \leq \pi(y)q(y,x)$, we allow the transition from x to y without restriction.
- If $\pi(x)q(x,y) > \pi(y)q(y,x)$, we allow the transition with probability $\alpha(x,y) = \min\left\{1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right\}$.

To summarize the two cases:

- If $\pi(x)q(x,y) \leq \pi(y)q(y,x)$, we allow the transition from x to y without restriction.
- If $\pi(x)q(x,y) > \pi(y)q(y,x)$, we allow the transition with probability $\alpha(x,y) = \min\left\{1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right\}$.

This acceptance function $\alpha(x,y)$ is the core idea behind the **Metropolis algorithm**.

Algorithm:

Algorithm:

The **Metropolis algorithm** proceeds as follows:

Algorithm:

The **Metropolis algorithm** proceeds as follows:

1. Start from an initial value x.

Algorithm:

The **Metropolis algorithm** proceeds as follows:

1. Start from an initial value x.
2. Propose a new state y by generating $y \sim Q(x, \cdot)$.

Algorithm:

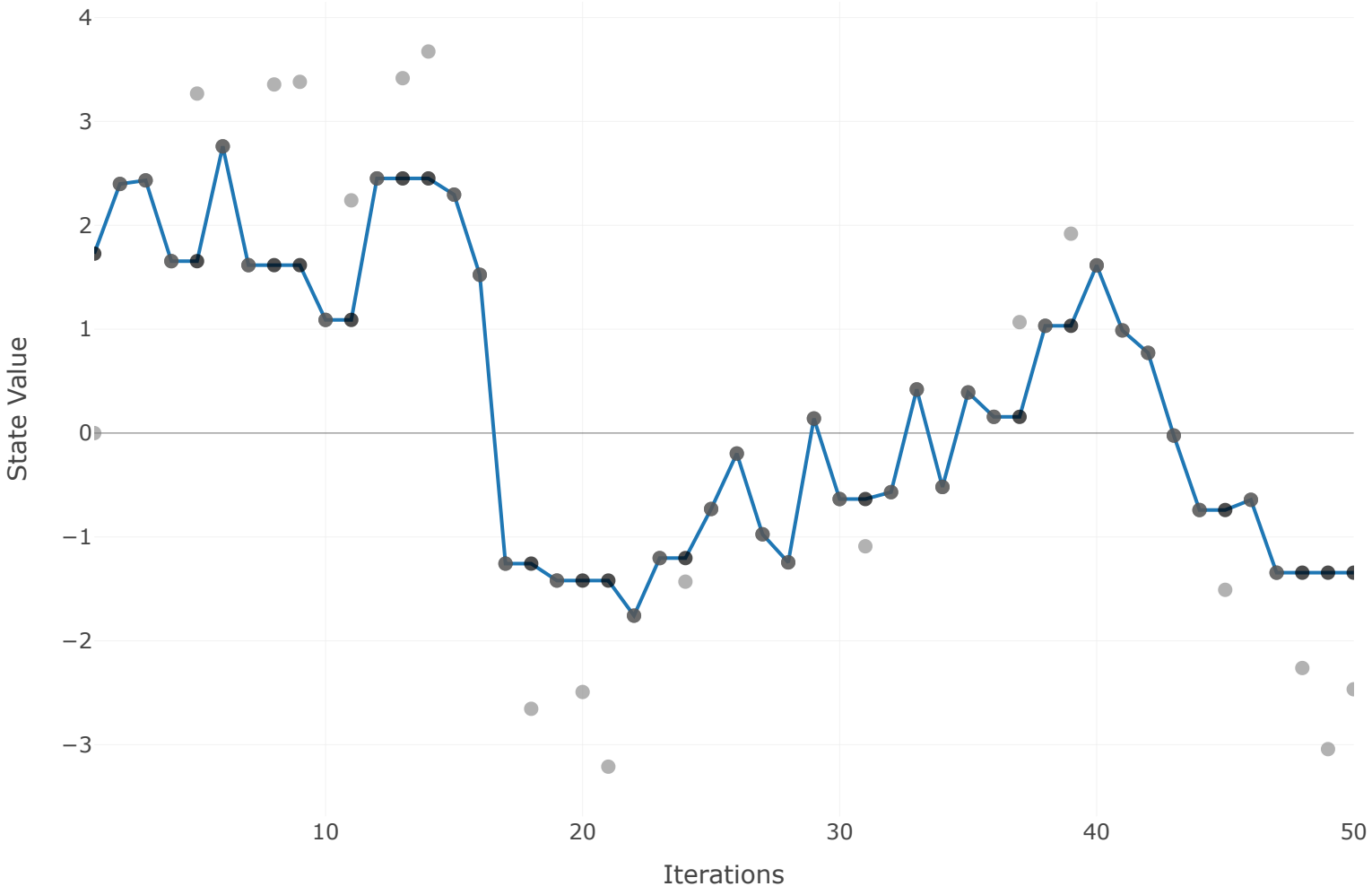
The **Metropolis algorithm** proceeds as follows:

1. Start from an initial value x.
2. Propose a new state y by generating $y \sim Q(x, \cdot)$.
3. Draw $u \sim \mathcal{U}[0, 1]$.

Algorithm:

The **Metropolis algorithm** proceeds as follows:

1. Start from an initial value x.
2. Propose a new state y by generating $y \sim Q(x, \cdot)$.
3. Draw $u \sim \mathcal{U}[0, 1]$.
4. If $u \leq \alpha(x, y)$, move to y, otherwise stay at x.


State Value
4
3
2
1
0
−1
−2
−3
10
20
30
40
50
Iterations

Reversiblity

> The **outflow** and **inflow** becomes the same. A Markov chain with this balance is called **reversible**. This prpoerty is the basis for the Metropolis algorithm.

Reversiblity

> The **outflow** and **inflow** becomes the same. A Markov chain with this balance is called **reversible**. This prpoerty is the basis for the Metropolis algorithm.

A Markov kernel $P(x, \cdot)$ is called Π-**reversible** for a measure Π if

$$\int_A \Pi(\mathrm{d}x) P(x, B) = \int_B \Pi(\mathrm{d}x) P(x, A)$$

are the same for any A and B.

Reversiblity

> The **outflow** and **inflow** becomes the same. A Markov chain with this balance is called **reversible**. This prpoerty is the basis for the Metropolis algorithm.

A Markov kernel $P(x, \cdot)$ is called Π-**reversible** for a measure Π if

$$\int_A \Pi(\mathrm{d}x) P(x, B) = \int_B \Pi(\mathrm{d}x) P(x, A)$$

are the same for any A and B.

If Π is a probability measure, then Π is an invariant probability measure.

Summary

Summary

- The **Markov Chain Monte Carlo (MCMC)** algorithm is a numerical method for approximating integrals, leveraging the law of large numbers for Markov chains.

Summary

- The **Markov Chain Monte Carlo (MCMC)** algorithm is a numerical method for approximating integrals, leveraging the law of large numbers for Markov chains.
- The **Metropolis algorithm**, a core MCMC technique, relies on the principle of reversibility.

Summary

- The **Markov Chain Monte Carlo (MCMC)** algorithm is a numerical method for approximating integrals, leveraging the law of large numbers for Markov chains.
- The **Metropolis algorithm**, a core MCMC technique, relies on the principle of reversibility.
- The Metropolis algorithm operates by repeating a two-step process: **Propose** a new state and then either **Accept** or **Reject** it.

2.2 Random Walk Metropolis Algorithm

2.2 Random Walk Metropolis Algorithm

> The **random walk Metropolis** algorithm is the earliest and one of the most well-known MCMC methods.

2.2 Random Walk Metropolis Algorithm

> The **random walk Metropolis** algorithm is the earliest and one of the most well-known MCMC methods.

The most basic type of Metropolis algorithm is the **random walk Metropolis algorithm**. In this method, a random step is proposed from the current state, and the step is either accepted or rejected based on the acceptance ratio.

2.2 Random Walk Metropolis Algorithm

> The **random walk Metropolis** algorithm is the earliest and one of the most well-known MCMC methods.

The most basic type of Metropolis algorithm is the **random walk Metropolis algorithm**. In this method, a random step is proposed from the current state, and the step is either accepted or rejected based on the acceptance ratio.

Here's an example of a random walk Metropolis algorithm in R:

2.2 Random Walk Metropolis Algorithm

> The **random walk Metropolis** algorithm is the earliest and one of the most well-known MCMC methods.

The most basic type of Metropolis algorithm is the **random walk Metropolis algorithm**. In this method, a random step is proposed from the current state, and the step is either accepted or rejected based on the acceptance ratio.

Here's an example of a random walk Metropolis algorithm in R:

```r
sd <- 0.25
target <- function(x) 1/(1 + x^2)

v <- numeric(1e2)
x <- runif(1)
for(i in 1:length(v)) {
  y <- x + sd * rnorm(1)
  if(runif(1) < target(y)/target(x)) x <- y #<< Acceptance step
  v[i] <- x
}
```

In this algorithm:

$$y = x + \sigma w, \quad w \sim \mathcal{N}(0, I_d),$$

where σ is a step size, and w is a random sample from the normal distribution.

In this algorithm:

$$y = x + \sigma w, \quad w \sim \mathcal{N}(0, I_d),$$

where σ is a step size, and w is a random sample from the normal distribution.

If the new state y is accepted, it becomes the next state of the Markov chain; otherwise, the chain remains at x.

Choice of σ and dimensionality

Choice of σ and dimensionality

> The selection of the standard deviation in the random-walk Metropolis algorithm has been widely discussed. In this section, we focus on **Roberts, Gelman and Gilks 97** , which provides valuable insight into how to make this choice.

Choice of σ and dimensionality

> The selection of the standard deviation in the random-walk Metropolis algorithm has been widely discussed. In this section, we focus on **Roberts, Gelman and Gilks 97** , which provides valuable insight into how to make this choice.

- The **Metropolis-Hastings algorithm** is affected by the **curse of dimensionality**, directly tied to the choice of the step size, σ.

Choice of σ and dimensionality

> The selection of the standard deviation in the random-walk Metropolis algorithm has been widely discussed. In this section, we focus on **Roberts, Gelman and Gilks 97** , which provides valuable insight into how to make this choice.

- The **Metropolis-Hastings algorithm** is affected by the **curse of dimensionality**, directly tied to the choice of the step size, σ.
- Balancing **outflow** and **inflow** (reversibility), can introduce challenges for the algorithm's efficiency.

Average acceptance probability

For a Markov chain generated by the random-walk Metropolis algorithm $(X_m)_m$, if $\|X_0\| \neq \|X_1\|$, it means the move at time $t = 1$ has been **accepted**.

Average acceptance probability

For a Markov chain generated by the random-walk Metropolis algorithm $(X_m)_m$, if $\|X_0\| \neq \|X_1\|$, it means the move at time $t = 1$ has been **accepted**.

Therefore, the probability of accepting a move is:

$$\mathbb{P}(\|X_0\| \neq \|X_1\|).$$

Average acceptance probability

For a Markov chain generated by the random-walk Metropolis algorithm $(X_m)_m$, if $\|X_0\| \neq \|X_1\|$, it means the move at time $t = 1$ has been **accepted**.

Therefore, the probability of accepting a move is:

$$\mathbb{P}(\|X_0\| \neq \|X_1\|).$$

Due to reversibility, we have the following balance property:

$$\mathbb{P}(\|X_0\|^2 < \|X_1\|^2) = \mathbb{P}(\|X_0\|^2 > \|X_1\|^2).$$

Average acceptance probability

For a Markov chain generated by the random-walk Metropolis algorithm $(X_m)_m$, if $\|X_0\| \neq \|X_1\|$, it means the move at time $t = 1$ has been **accepted**.

Therefore, the probability of accepting a move is:

$$\mathbb{P}(\|X_0\| \neq \|X_1\|).$$

Due to reversibility, we have the following balance property:

$$\mathbb{P}(\|X_0\|^2 < \|X_1\|^2) = \mathbb{P}(\|X_0\|^2 > \|X_1\|^2).$$

Thus, the acceptance probability equals:

$$\mathbf{2} \cdot \mathbb{P}(\|X_0\|^2 > \|X_1\|^2).$$

Average acceptance probability

For a Markov chain generated by the random-walk Metropolis algorithm $(X_m)_m$, if $\|X_0\| \neq \|X_1\|$, it means the move at time $t = 1$ has been **accepted**.

Therefore, the probability of accepting a move is:

$$\mathbb{P}(\|X_0\| \neq \|X_1\|).$$

Due to reversibility, we have the following balance property:

$$\mathbb{P}(\|X_0\|^2 < \|X_1\|^2) = \mathbb{P}(\|X_0\|^2 > \|X_1\|^2).$$

Thus, the acceptance probability equals:

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_1\|^2).$$

Denote the proposed state as Y_1, then the acceptance probability is **less than**:

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2).$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

Then ...

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2) = 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0 + \sigma W\|^2)$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

Then ...

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2) = 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0 + \sigma W\|^2)$$

$$= 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0\|^2 + 2\sigma X_0^\top W + \sigma^2 \|W\|^2)$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

Then ...

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2) = 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0 + \sigma W\|^2)$$

$$= 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0\|^2 + 2\sigma X_0^\top W + \sigma^2 \|W\|^2)$$

$$= 2 \cdot \mathbb{P}(-2\sigma X_0^\top W > \sigma^2 \|W\|^2)$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

Then ...

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2) = 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0 + \sigma W\|^2)$$

$$= 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0\|^2 + 2\sigma X_0^\top W + \sigma^2 \|W\|^2)$$

$$= 2 \cdot \mathbb{P}(-2\sigma X_0^\top W > \sigma^2 \|W\|^2)$$

$$= 2 \cdot \mathbb{P}(-2X_0^\top e > \sigma \|W\|),$$

The proposed state is generated as follows:

$$Y_1 = X_0 + \sigma W, \quad W \sim \mathcal{N}_d(0, \sigma I_d).$$

Then ...

$$2 \cdot \mathbb{P}(\|X_0\|^2 > \|Y_1\|^2) = 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0 + \sigma W\|^2)$$

$$= 2 \cdot \mathbb{P}(\|X_0\|^2 > \|X_0\|^2 + 2\sigma X_0^\top W + \sigma^2 \|W\|^2)$$

$$= 2 \cdot \mathbb{P}(-2\sigma X_0^\top W > \sigma^2 \|W\|^2)$$

$$= 2 \cdot \mathbb{P}(-2 X_0^\top e > \sigma \|W\|),$$

where $e = W/\|W\|$ follows a uniform distribution on the unit sphere.

Dimensionality Effects

- Assuming $X_0^\top e = O(1)$ and $\|W\| = O(d^{1/2})$, unless $\sigma = O(d^{-1/2})$, the acceptance probability **decreases quickly**.

Dimensionality Effects

- Assuming $X_0^\top e = O(1)$ and $\|W\| = O(d^{1/2})$, unless $\sigma = O(d^{-1/2})$, the acceptance probability **decreases quickly**.
- Conversely, if $\sigma = O(d^{-1/2})$, the algorithm cannot **make large moves**, revealing the curse of dimensionality.

Taking it Further

If X_0 follows a d-dimensional standard normal distribution, the right-hand side becomes:

$$2\mathbb{E}\left[\Phi(-\sigma\|W\|/2)\right].$$

Taking it Further

If X_0 follows a d-dimensional standard normal distribution, the right-hand side becomes:

$$2\mathbb{E}\left[\Phi(-\sigma\|W\|/2)\right].$$

A useful performance measure for the random-walk Metropolis algorithm is the **expected squared jump distance (ESJD)**:

$$\mathbb{E}\left[\|X_1 - X_0\|^2\right].$$

> There are few reliable measures for assessing the performance of MCMC methods. ESJD is sometimes used because it is relatively straightforward to analyze theoretically. However, like many performance metrics, the direct link between ESJD and actual algorithm efficiency remains unclear.

ESJD can be simplified as:

$$2\mathbb{E}\left[\|X_1 - X_0\|^2, \|X_0\|^2 > \|X_1\|^2\right] = \mathbb{E}\left[\sigma^2\|W\|^2, -2X_0^\top e > \sigma\|W\|\right].$$

ESJD can be simplified as:

$$2\mathbb{E}\left[\|X_1 - X_0\|^2, \|X_0\|^2 > \|X_1\|^2\right] = \mathbb{E}\left[\sigma^2\|W\|^2, -2X_0^\top e > \sigma\|W\|\right].$$

$$= \mathbb{E}\left[\Phi(-\sigma\|W\|/2) \cdot \sigma^2\|W\|^2\right].$$

ESJD can be simplified as:

$$2\mathbb{E}\left[\|X_1 - X_0\|^2, \|X_0\|^2 > \|X_1\|^2\right] = \mathbb{E}\left[\sigma^2\|W\|^2, -2X_0^\top e > \sigma\|W\|\right].$$

$$= \mathbb{E}\left[\Phi(-\sigma\|W\|/2) \cdot \sigma^2\|W\|^2\right].$$

This value is maximized when $\sigma\|W\|$ takes a specific value. At this maximum, the average acceptance probability is approximately **23.4%** (treating $\sigma\|W\|$ as constant).

ESJD can be simplified as:

$$2\mathbb{E}\left[\|X_1 - X_0\|^2, \|X_0\|^2 > \|X_1\|^2\right] = \mathbb{E}\left[\sigma^2\|W\|^2, -2X_0^\top e > \sigma\|W\|\right].$$

$$= \mathbb{E}\left[\Phi(-\sigma\|W\|/2) \cdot \sigma^2\|W\|^2\right].$$

This value is maximized when $\sigma\|W\|$ takes a specific value. At this maximum, the average acceptance probability is approximately **23.4%** (treating $\sigma\|W\|$ as constant).

So we may use **23.4%** ratio as a criterion of the choice of the tuning parameter σ.

ESJD can be simplified as:

$$2\mathbb{E}\left[\|X_1 - X_0\|^2, \|X_0\|^2 > \|X_1\|^2\right] = \mathbb{E}\left[\sigma^2\|W\|^2, -2X_0^\top e > \sigma\|W\|\right].$$

$$= \mathbb{E}\left[\Phi(-\sigma\|W\|/2) \cdot \sigma^2\|W\|^2\right].$$

This value is maximized when $\sigma\|W\|$ takes a specific value. At this maximum, the average acceptance probability is approximately **23.4%** (treating $\sigma\|W\|$ as constant).

So we may use **23.4%** ratio as a criterion of the choice of the tuning parameter σ.

We initially assumed $X_0 \sim \mathcal{N}(0, I_d)$, meaning $\Pi = \mathcal{N}(0, I_d)$. However, this assumption can be relaxed. Empirically, it is believed that this criterion is relatively robust across different settings.

Summary

Summary

- The choice of the step size σ is crucial for the performance of the algorithm. If σ is too large, the rejection rate increases. If σ is too small, the proposed state will be too close to the current one, leading to slow exploration.

Summary

- The choice of the step size σ is crucial for the performance of the algorithm. If σ is too large, the rejection rate increases. If σ is too small, the proposed state will be too close to the current one, leading to slow exploration.
- This impact can be quantified using the **expected squared jump distance (ESJD)**, a measure of how far the chain moves.

Summary

- The choice of the step size σ is crucial for the performance of the algorithm. If σ is too large, the rejection rate increases. If σ is too small, the proposed state will be too close to the current one, leading to slow exploration.
- This impact can be quantified using the **expected squared jump distance (ESJD)**, a measure of how far the chain moves.
- There is an optimal value of σ that maximizes the ESJD. When this is achieved, the average acceptance rate is approximately **23.4%**.

Exercise 3.

Exercise 3.

3.1 Verify that

$$\alpha(x, y) = \frac{\pi(y)\, q(y, x)}{\pi(x)\, q(x, y) + \pi(y)\, q(y, x)}$$

also satisfies the balance between **inflow** and **outflow**.

Exercise 3.

3.1 Verify that

$$\alpha(x, y) = \frac{\pi(y)\, q(y, x)}{\pi(x)\, q(x, y) + \pi(y)\, q(y, x)}$$

also satisfies the balance between **inflow** and **outflow**.

3.2 Write a code (R, python etc.) for a random walk Metropolis algorithm for the distribution $\pi(x) \propto \frac{1}{1+x^2}$.

Exercise 3.

3.1 Verify that

$$\alpha(x, y) = \frac{\pi(y)\, q(y, x)}{\pi(x)\, q(x, y) + \pi(y)\, q(y, x)}$$

also satisfies the balance between **inflow** and **outflow**.

3.2 Write a code (R, python etc.) for a random walk Metropolis algorithm for the distribution $\pi(x) \propto \frac{1}{1+x^2}$.

3.3 Discuss the use of a heavy-tailed distribution as the law of W in terms of the Expected Squared Jump Distance when the target distribution is standard normal.

2.3 Preconditioned Crank-Nicolson (pCN) and Gaussian perturbation

2.3 Preconditioned Crank-Nicolson (pCN) and Gaussian perturbation

> The pCN algorithm uses an autoregressive model as its proposal mechanism. This approach is particularly effective when the target distribution is a small perturbation of a normal distribution.

2.3 Preconditioned Crank-Nicolson (pCN) and Gaussian perturbation

> The pCN algorithm uses an autoregressive model as its proposal mechanism. This approach is particularly effective when the target distribution is a small perturbation of a normal distribution.

A variant of the Metropolis algorithm is the **preconditioned Crank-Nicolson (pCN)** method, often used in Bayesian computation. This method is well-suited for high-dimensional problems.

2.3 Preconditioned Crank-Nicolson (pCN) and Gaussian perturbation

> The pCN algorithm uses an autoregressive model as its proposal mechanism. This approach is particularly effective when the target distribution is a small perturbation of a normal distribution.

A variant of the Metropolis algorithm is the **preconditioned Crank-Nicolson (pCN)** method, often used in Bayesian computation. This method is well-suited for high-dimensional problems.

The proposed state in the pCN algorithm is given by:

$$y = \rho x + \sigma\sqrt{1-\rho^2}w, \quad w \sim \mathcal{N}(0, I_d),$$

where $\rho \in (-1, 1)$ is a tuning parameter.

Bayesian Inverse Problem

In Bayesian statistics, we are concerned with the posterior distribution, which is defined as:

$$\Pi(\mathrm{d}\theta) = \frac{\exp(l(\theta))P(\mathrm{d}\theta)}{\int_\Theta \exp(l(\theta))P(\mathrm{d}\theta)},$$

where $P(\mathrm{d}\theta)$ is the prior distribution and $l(\theta)$ represents the log-likelihood.

Bayesian Inverse Problem

In Bayesian statistics, we are concerned with the posterior distribution, which is defined as:

$$\Pi(\mathrm{d}\theta) = \frac{\exp(l(\theta))P(\mathrm{d}\theta)}{\int_\Theta \exp(l(\theta))P(\mathrm{d}\theta)},$$

where $P(\mathrm{d}\theta)$ is the prior distribution and $l(\theta)$ represents the log-likelihood.

The goal is to compute the expectation of a function with respect to this posterior distribution.

Bayesian Inverse Problem

In Bayesian statistics, we are concerned with the posterior distribution, which is defined as:

$$\Pi(\mathrm{d}\theta) = \frac{\exp(l(\theta))P(\mathrm{d}\theta)}{\int_\Theta \exp(l(\theta))P(\mathrm{d}\theta)},$$

where $P(\mathrm{d}\theta)$ is the prior distribution and $l(\theta)$ represents the log-likelihood.

The goal is to compute the expectation of a function with respect to this posterior distribution.

In typical Bayesian inverse problems, the parameter space is a high-dimensional Euclidean space, with $P(\mathrm{d}\theta)$ being a normal distribution.

Bayesian Inverse Problem

In Bayesian statistics, we are concerned with the posterior distribution, which is defined as:

$$\Pi(\mathrm{d}\theta) = \frac{\exp(l(\theta))P(\mathrm{d}\theta)}{\int_\Theta \exp(l(\theta))P(\mathrm{d}\theta)},$$

where $P(\mathrm{d}\theta)$ is the prior distribution and $l(\theta)$ represents the log-likelihood.

The goal is to compute the expectation of a function with respect to this posterior distribution.

In typical Bayesian inverse problems, the parameter space is a high-dimensional Euclidean space, with $P(\mathrm{d}\theta)$ being a normal distribution.

In this setting, the random-walk Metropolis algorithm tends to perform poorly. But how does the pCN algorithm fare?

Bayesian Inverse Problem

In Bayesian statistics, we are concerned with the posterior distribution, which is defined as:

$$\Pi(\mathrm{d}\theta) = \frac{\exp(l(\theta))P(\mathrm{d}\theta)}{\int_\Theta \exp(l(\theta))P(\mathrm{d}\theta)},$$

where $P(\mathrm{d}\theta)$ is the prior distribution and $l(\theta)$ represents the log-likelihood.

The goal is to compute the expectation of a function with respect to this posterior distribution.

In typical Bayesian inverse problems, the parameter space is a high-dimensional Euclidean space, with $P(\mathrm{d}\theta)$ being a normal distribution.

In this setting, the random-walk Metropolis algorithm tends to perform poorly. But how does the pCN algorithm fare?

Assume $P = \mathcal{N}(0, 1)$ and let $\sigma = 1$.

Thanks to the autoregressive nature of the pCN model, the transition:

$$y = \rho x + \sqrt{1-\rho^2} w, \quad w \sim \mathcal{N}(0, I_d),$$

is **reversible** with respect to $\mathcal{N}(0, I_d)$.

Thanks to the autoregressive nature of the pCN model, the transition:

$$y = \rho x + \sqrt{1-\rho^2} w, \quad w \sim \mathcal{N}(0, I_d),$$

is **reversible** with respect to $\mathcal{N}(0, I_d)$.

This means that only minimal **traffic control** is required for ensuring stability.

Thanks to the autoregressive nature of the pCN model, the transition:

$$y = \rho x + \sqrt{1-\rho^2} w, \quad w \sim \mathcal{N}(0, I_d),$$

is **reversible** with respect to $\mathcal{N}(0, I_d)$.

This means that only minimal **traffic control** is required for ensuring stability.

Specifically, the acceptance probability is given by:

$$\min\left\{1, \exp(l(\theta^*) - l(\theta))\right\},$$

where θ^* is the proposed state and θ is the current state.

Thanks to the autoregressive nature of the pCN model, the transition:

$$y = \rho x + \sqrt{1-\rho^2} w, \quad w \sim \mathcal{N}(0, I_d),$$

is **reversible** with respect to $\mathcal{N}(0, I_d)$.

This means that only minimal **traffic control** is required for ensuring stability.

Specifically, the acceptance probability is given by:

$$\min\left\{1, \exp(l(\theta^*) - l(\theta))\right\},$$

where θ^* is the proposed state and θ is the current state.

Even in high dimensions, sometimes this ratio tends to be stable, unlike the behavior seen with the random-walk Metropolis algorithm.

Summary

- The pCN algorithm is specifically designed for handling Gaussian perturbations.

Summary

- The pCN algorithm is specifically designed for handling Gaussian perturbations.
- In high-dimensional settings, it often maintains a high acceptance probability.

Summary

- The pCN algorithm is specifically designed for handling Gaussian perturbations.
- In high-dimensional settings, it often maintains a high acceptance probability.
- However, it's important to note that its performance deteriorates significantly when the target distribution deviates far from Gaussian. This method is sometimes not robust.

Exercise 4.

Exercise 4.

4.1 Verify that the acceptance ratio for the preconditioned Crank-Nicolson algorithm, with the target distribution

$$\Pi(\mathrm{d}x) \propto \exp(-U(x) - x^2/2)\mathrm{d}x$$

is given by

$$\alpha(x, y) = \min\left\{1, \exp(-U(y) + U(x))\right\}.$$

Exercise 4.

4.1 Verify that the acceptance ratio for the preconditioned Crank-Nicolson algorithm, with the target distribution

$$\Pi(\mathrm{d}x) \propto \exp(-U(x) - x^2/2)\mathrm{d}x$$

is given by

$$\alpha(x, y) = \min\left\{1, \exp(-U(y) + U(x))\right\}.$$

4.2 Write code (in R, Python, etc.) for the preconditioned Crank-Nicolson algorithm targeting the distribution $\pi(x) \propto \frac{1}{1+x^2}$. Conclude that the performance, although not mathematically defined, is worse compared to the random walk Metropolis algorithm.

Exercise 4.

4.1 Verify that the acceptance ratio for the preconditioned Crank-Nicolson algorithm, with the target distribution

$$\Pi(\mathrm{d}x) \propto \exp(-U(x) - x^2/2)\mathrm{d}x$$

is given by

$$\alpha(x, y) = \min\left\{1, \exp(-U(y) + U(x))\right\}.$$

4.2 Write code (in R, Python, etc.) for the preconditioned Crank-Nicolson algorithm targeting the distribution $\pi(x) \propto \frac{1}{1+x^2}$. Conclude that the performance, although not mathematically defined, is worse compared to the random walk Metropolis algorithm.

4.3 Discuss the reason behind this observation.

3. (Advanced) Non-reversiblility

3.1 Non-Reversibility in descrete time

3.1 Non-Reversibility in descrete time

> From here, we'll explore non-reversible methods as an advanced topic in this lecture. Although designing non-reversible algorithms is technically challenging, they offer a broader range of options and can enhance the efficiency of certain sampling techniques.

3.1 Non-Reversibility in descrete time

> From here, we'll explore non-reversible methods as an advanced topic in this lecture. Although designing non-reversible algorithms is technically challenging, they offer a broader range of options and can enhance the efficiency of certain sampling techniques.

Let’s now discuss the **reversibility** and **non-reversibility** in the Metropolis algorithms.

3.1 Non-Reversibility in descrete time

> From here, we'll explore non-reversible methods as an advanced topic in this lecture. Although designing non-reversible algorithms is technically challenging, they offer a broader range of options and can enhance the efficiency of certain sampling techniques.

Let’s now discuss the **reversibility** and **non-reversibility** in the Metropolis algorithms.

Reversibility simplifies algorithm design but can sometimes lead to inefficiency.

3.1 Non-Reversibility in descrete time

> From here, we'll explore non-reversible methods as an advanced topic in this lecture. Although designing non-reversible algorithms is technically challenging, they offer a broader range of options and can enhance the efficiency of certain sampling techniques.

Let's now discuss the **reversibility** and **non-reversibility** in the Metropolis algorithms.

Reversibility simplifies algorithm design but can sometimes lead to inefficiency.

In certain cases, the Markov chain spends too much time exploring the same areas, leading to longer travel distances before covering the state space adequately.

Lifting

Lifting

One way to introduce non-reversibility is through a technique called **lifting**, which involves extending the state space.

Lifting

One way to introduce non-reversibility is through a technique called **lifting**, which involves extending the state space.

In lifting, we introduce a velocity variable $v \in \{-1, +1\}$ and allow the chain to switch directions at certain points.

Lifting

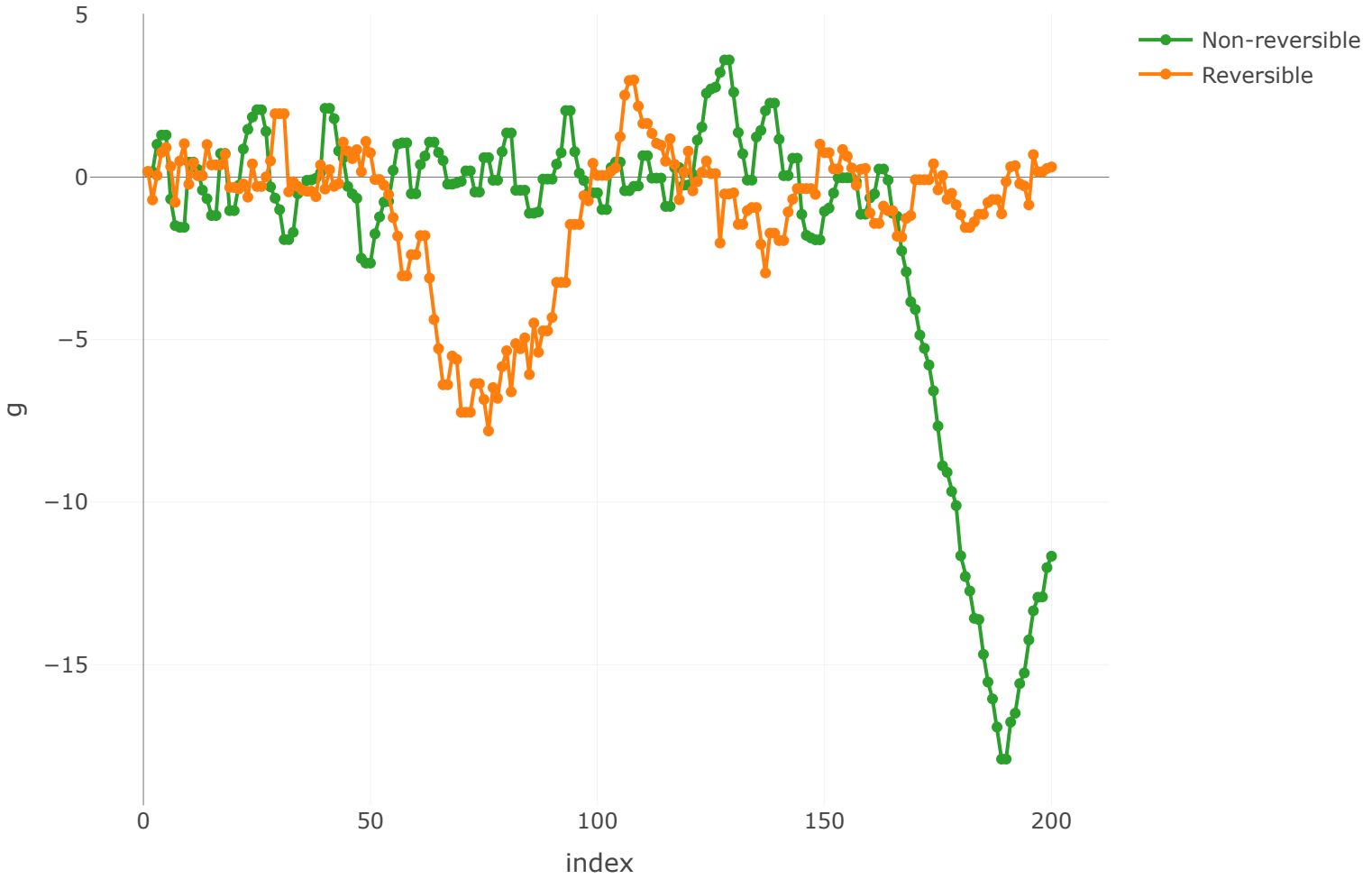
One way to introduce non-reversibility is through a technique called **lifting**, which involves extending the state space.

In lifting, we introduce a velocity variable $v \in \{-1, +1\}$ and allow the chain to switch directions at certain points.

For example, starting from $(x, +1)$, the algorithm can move to $(x + |w|, -1)$ if the proposal is accepted. If rejected, it stays at $(x, +1)$.

Lifting Visualization: Example by Gustafson 1996

Here’s an example of how lifting works in practice:



Summary

Summary

- Recall that the Metropolis algorithm is a two-step procedure: propose and then accept or reject.

Summary

- Recall that the Metropolis algorithm is a two-step procedure: propose and then accept or reject.
- The lifting procedure works as follows:

Summary

- Recall that the Metropolis algorithm is a two-step procedure: propose and then accept or reject.
- The lifting procedure works as follows:
- If the proposal is accepted, the direction remains unchanged. If it is rejected, the sign is reversed.

Summary

- Recall that the Metropolis algorithm is a two-step procedure: propose and then accept or reject.
- The lifting procedure works as follows:
- If the proposal is accepted, the direction remains unchanged. If it is rejected, the sign is reversed.

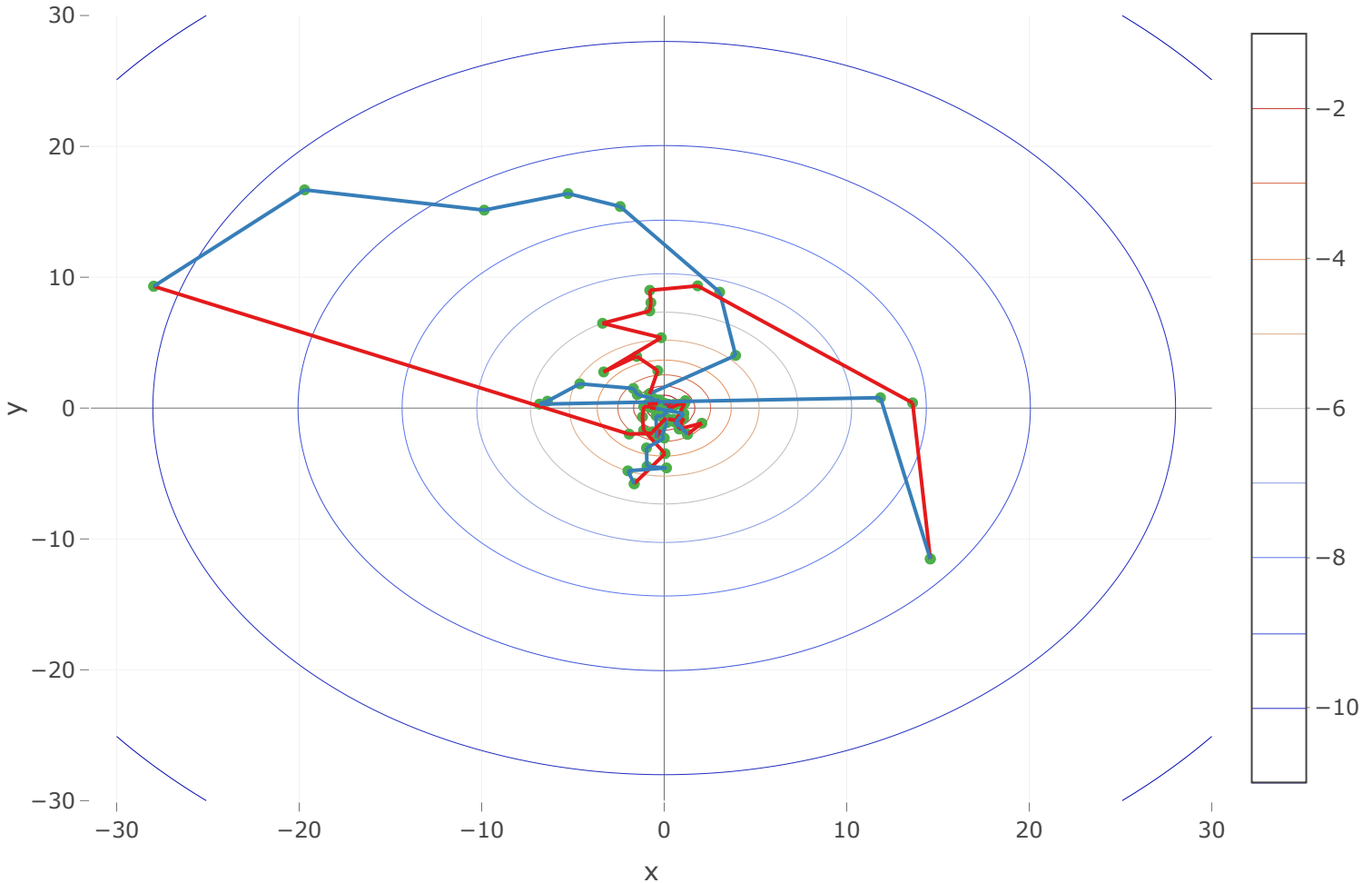
This method works well in one dimension. However, extending it to multiple dimensions requires special care.

Some recent progress in puctures (K., Song 2023)

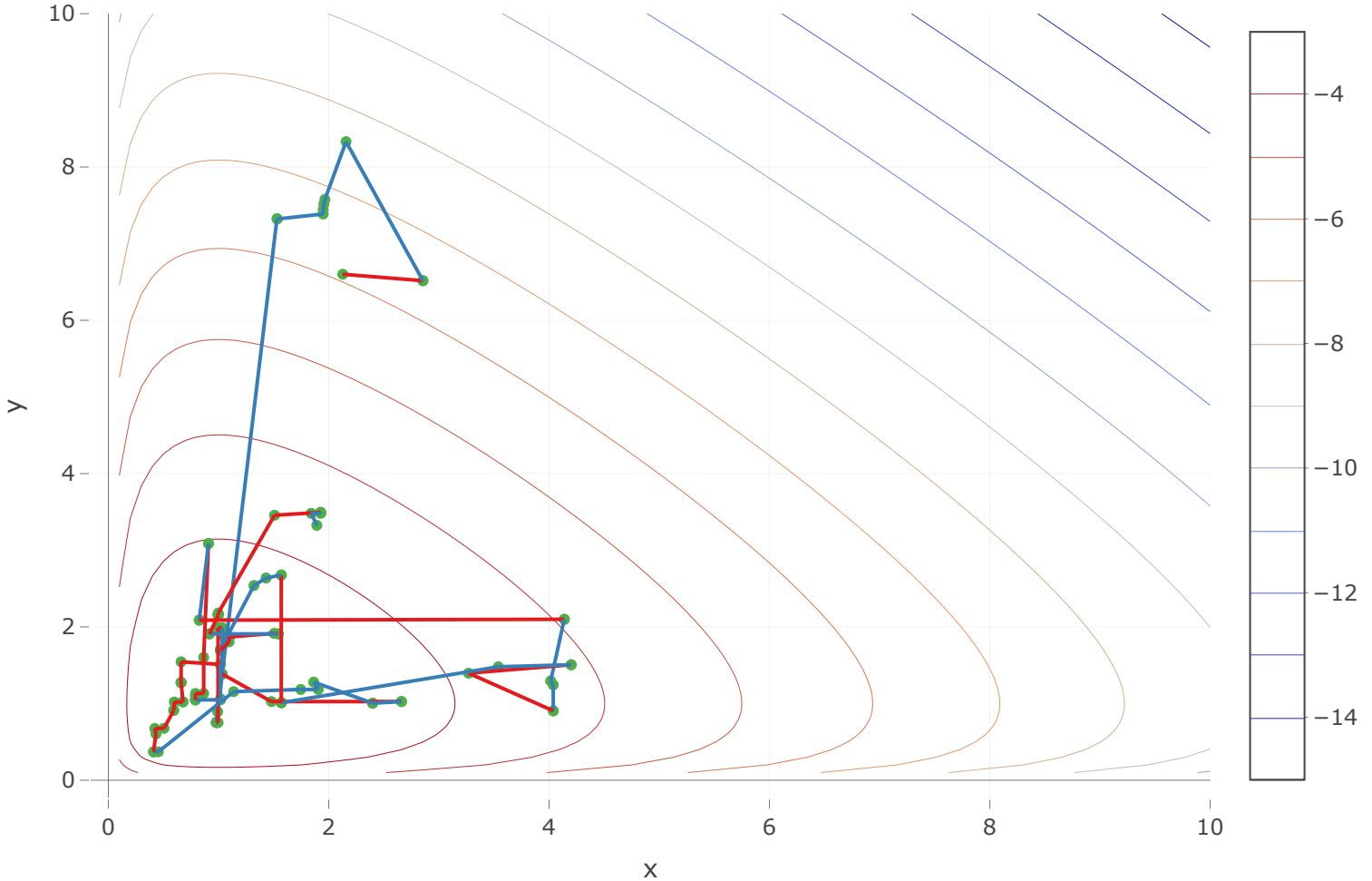
Some recent progress in puctures (K., Song 2023)

> Extending the previous method to the multidimensional case in a more useful direction was challenging. Due to time constraints, we present the recent results in pictures only.

Multi-dim case: $\mathbb{R}^d$ (Autoregressive)



Multi-dim case: $\mathbb{R}_+^d$ (Beta-Gamma)



Exercise 5.

Exercise 5.

5.1 Write code (in R, Python, etc.) for the random walk Metropolis algorithm with lifting with targeting the distribution $\pi(x) \propto \frac{1}{1+x^2}$. Plot a path and compare it to that of the random walk Metropolis algorithm.

3.2 Non-reversibility in continous time

3.2 Non-reversibility in continous time

> Designing non-reversible methods has been challenging because it requires balancing **inflow** and **outflow** in many possible ways. However, if we focus on continuous time processes, things become simpler, as the behavior is essentially characterized around each point.

3.2 Non-reversibility in continous time

> Designing non-reversible methods has been challenging because it requires balancing **inflow** and **outflow** in many possible ways. However, if we focus on continuous time processes, things become simpler, as the behavior is essentially characterized around each point.

Recently, algorithms that operate in continuous time, such as the **Zig-Zag Sampler** and **Bouncy Particle Sampler**, have gained attention.

3.2 Non-reversibility in continous time

> Designing non-reversible methods has been challenging because it requires balancing **inflow** and **outflow** in many possible ways. However, if we focus on continuous time processes, things become simpler, as the behavior is essentially characterized around each point.

Recently, algorithms that operate in continuous time, such as the **Zig-Zag Sampler** and **Bouncy Particle Sampler**, have gained attention.

These algorithms define **piecewise determinsitic Markov processes**.

3.2 Non-reversibility in continous time

> Designing non-reversible methods has been challenging because it requires balancing **inflow** and **outflow** in many possible ways. However, if we focus on continuous time processes, things become simpler, as the behavior is essentially characterized around each point.

Recently, algorithms that operate in continuous time, such as the **Zig-Zag Sampler** and **Bouncy Particle Sampler**, have gained attention.

These algorithms define **piecewise determinsitic Markov processes**.

These methods remove the need to define discrete-time steps and allow the chain to move continuously through the state space.

A piecewise deterministic Markov process (PDMP) is a Markov process consisting of a **drift term** and a **Jump term**, without a diffusion term.

A piecewise deterministic Markov process (PDMP) is a Markov process consisting of a **drift term** and a **Jump term**, without a diffusion term.

This process has been highlighted in Monte Carlo literature, specifically in **Peters and de With 2012** and **Michel et al. 2014** .

A piecewise deterministic Markov process (PDMP) is a Markov process consisting of a **drift term** and a **Jump term**, without a diffusion term.

This process has been highlighted in Monte Carlo literature, specifically in **Peters and de With 2012** and **Michel et al. 2014** .

The PDMP is characterised by being **non-reversible** and distinct from traditional Markov chain Monte Carlo (MCMC) methods, as it is a continuous-time process.

A piecewise deterministic Markov process (PDMP) is a Markov process consisting of a **drift term** and a **Jump term**, without a diffusion term.

This process has been highlighted in Monte Carlo literature, specifically in **Peters and de With 2012** and **Michel et al. 2014** .

The PDMP is characterised by being **non-reversible** and distinct from traditional Markov chain Monte Carlo (MCMC) methods, as it is a continuous-time process.

Also PDMPs seem to be suitable for a sub-sampling implementation.

PDMP triplet Davis1984

Let E be the state space. The triplet (φ, λ, Q) consists of:

PDMP triplet Davis1984

Let E be the state space. The triplet (φ, λ, Q) consists of:

- **Flow**: $(t, x) \mapsto \varphi_t(x)$, where $\varphi_0(x) = x$ and $\varphi_t(\varphi_s(x)) = \varphi_{t+s}(x)$.

PDMP triplet Davis1984

Let E be the state space. The triplet (φ, λ, Q) consists of:

- **Flow**: $(t, x) \mapsto \varphi_t(x)$, where $\varphi_0(x) = x$ and $\varphi_t(\varphi_s(x)) = \varphi_{t+s}(x)$.
- **Conditional intensity**: $\lambda(x) \geq 0$.

PDMP triplet Davis1984

Let E be the state space. The triplet (φ, λ, Q) consists of:

- **Flow**: $(t, x) \mapsto \varphi_t(x)$, where $\varphi_0(x) = x$ and $\varphi_t(\varphi_s(x)) = \varphi_{t+s}(x)$.
- **Conditional intensity**: $\lambda(x) \geq 0$.
- **Jump size**: $Q(x, A)$, which is a Markov kernel.

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_t(x) \quad (0 < t < T_1), \quad x(T_1) \sim Q(x(T_1-), \cdot)$$

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_t(x) \quad (0 < t < T_1), \quad x(T_1) \sim Q(x(T_1-), \cdot)$$

This determines $x(t)$ up to time T_1.

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_t(x) \quad (0 < t < T_1), \quad x(T_1) \sim Q(x(T_1-), \cdot)$$

This determines $x(t)$ up to time T_1.

Next, we repeat the following for $n = 2, 3, 4, \ldots$:

$$E_n \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_n = \inf\left\{t > T_{n-1} : E_n \leq \int_{T_{n-1}}^t \lambda(\varphi_{s-T_{n-1}}(x(T_{n-1})))\mathrm{d}s\right\}$$

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_t(x) \quad (0 < t < T_1), \quad x(T_1) \sim Q(x(T_1-), \cdot)$$

This determines $x(t)$ up to time T_1.

Next, we repeat the following for $n = 2, 3, 4, \ldots$:

$$E_n \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_n = \inf\left\{t > T_{n-1} : E_n \leq \int_{T_{n-1}}^t \lambda(\varphi_{s-T_{n-1}}(x(T_{n-1})))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_{t-T_{n-1}}(x(T_{n-1})) \quad (T_{n-1} < t < T_n), \quad x(T_n) \sim Q(x(T_n-), \cdot)$$

$$E_1 \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_1 = \inf\left\{t > 0 : E_1 \leq \int_0^t \lambda(\varphi_s(x))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_t(x) \quad (0 < t < T_1), \quad x(T_1) \sim Q(x(T_1-), \cdot)$$

This determines $x(t)$ up to time T_1.

Next, we repeat the following for $n = 2, 3, 4, \ldots$:

$$E_n \sim \mathcal{E}(1) \quad \rightsquigarrow \quad T_n = \inf\left\{t > T_{n-1} : E_n \leq \int_{T_{n-1}}^t \lambda(\varphi_{s-T_{n-1}}(x(T_{n-1})))\mathrm{d}s\right\}$$

$$\rightsquigarrow \quad x(t) = \varphi_{t-T_{n-1}}(x(T_{n-1})) \quad (T_{n-1} < t < T_n), \quad x(T_n) \sim Q(x(T_n-), \cdot)$$

This determines $x(t)$.

[1]

[1] If $\mathbb{P}(\sup_n T_n = \infty) \neq 1$ (i.e., if the process is non-explosive), then $x(t)$ is not defined for $t \in \mathbb{R}_+$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector
- Flow: v fixed. $x' = v$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector
- Flow: v fixed. $x' = v$.

There are are two souces of jumps:

- Conditional intensity: $\lambda(x) = (v^\top \nabla U(x))^+$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector
- Flow: v fixed. $x' = v$.

There are are two souces of jumps:

- Conditional intensity: $\lambda(x) = (v^\top \nabla U(x))^+$.
- Jump size: x fixed. $v \mapsto v - 2 \times v^\top n(x)\, n(x)$.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector
- Flow: v fixed. $x' = v$.

There are are two souces of jumps:

- Conditional intensity: $\lambda(x) = (v^\top \nabla U(x))^+$.
- Jump size: x fixed. $v \mapsto v - 2 \times v^\top n(x)\, n(x)$.

and

- Conditional intensity: ρ.

Example Bouncy particle sampler Peters and de With (2012)

The process is defined in $\mathbb{R}^d \times \mathbb{R}^d$.

We are interested in $\Pi(\mathrm{d}x) = \exp(-U(x))\mathrm{d}x$. Let $n(x) = \nabla U(x)/|\nabla U(x)|$.

- x: State variable
- v: Velocity vector
- Flow: v fixed. $x' = v$.

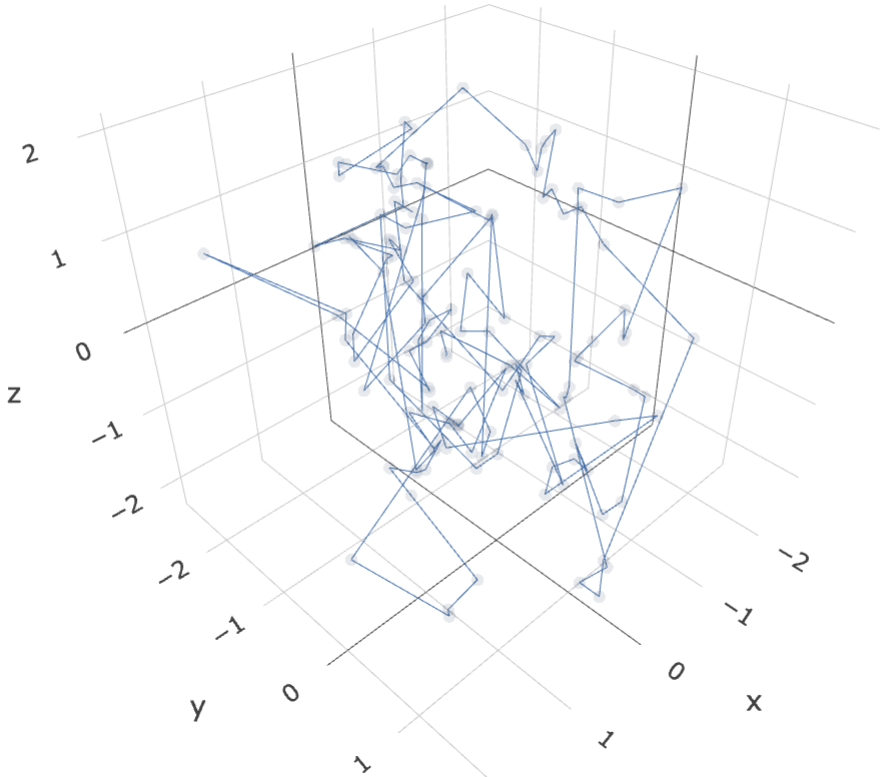
There are are two souces of jumps:

- Conditional intensity: $\lambda(x) = (v^\top \nabla U(x))^+$.
- Jump size: x fixed. $v \mapsto v - 2 \times v^\top n(x)\, n(x)$.

and

- Conditional intensity: ρ.
- Jump size: x fixed. $v \sim \mathcal{N}(0, I_d)$.

Here’s an example in three dimensions:



Extended generator

Extended generator

> A continuous Markov process is characterized by the **short-term behavior** of the process. For a Markov process (Z_t), this is essentially the derivative of $\mathbb{E}[f(Z_t) \mid Z_0 = z]$ with respect to t at $t = 0$.

Extended generator

> A continuous Markov process is characterized by the **short-term behavior** of the process. For a Markov process (Z_t), this is essentially the derivative of $\mathbb{E}[f(Z_t) \mid Z_0 = z]$ with respect to t at $t = 0$.

$$\mathcal{L}f(x, v) = v^\top \partial_x f(x, v) + (v^\top \nabla U(x))^+ \, (\mathcal{B} - \mathrm{id}) f(x, v) + \rho \, (\mathcal{R} - \mathrm{id}) f(x, v)$$

where $\mathcal{B}f(x, v) = f(x, B(x)v)$ and $B(x)v = (I - 2n(x)n(x)^\top)v$, and $\mathcal{R}$ is the **refresh** operator which changes the direction v randomly.

Example Zig-Zag sampler Bierkens, Fearnhead, and Roberts (2019)

Example Zig-Zag sampler Bierkens, Fearnhead, and Roberts (2019)

The process is defined in $\mathbb{R}^d \times \{-1, +1\}^d$.

Example Zig-Zag sampler Bierkens, Fearnhead, and Roberts (2019)

The process is defined in $\mathbb{R}^d \times \{-1, +1\}^d$.

- Flow: v fixed. $x' = v$.

Example Zig-Zag sampler Bierkens, Fearnhead, and Roberts (2019)

The process is defined in $\mathbb{R}^d \times \{-1, +1\}^d$.

- Flow: v fixed. $x' = v$.

There are d-sources of jumps:

- Conditional intensity: $\lambda_i(x) = (v_i \partial_i U(x))^+$.

Example Zig-Zag sampler Bierkens, Fearnhead, and Roberts (2019)

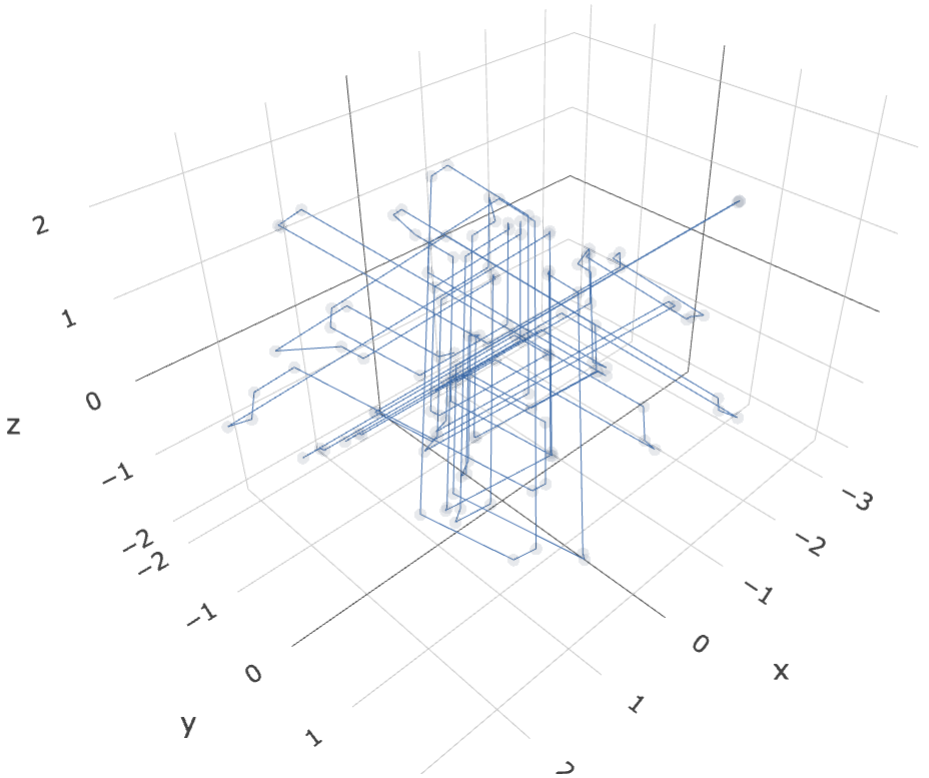
The process is defined in $\mathbb{R}^d \times \{-1, +1\}^d$.

- Flow: v fixed. $x' = v$.

There are d-sources of jumps:

- Conditional intensity: $\lambda_i(x) = (v_i \partial_i U(x))^+$.
- Jump size: x fixed. Switch the sign of i-th component of v.

Here’s an example of the Zig-Zag Sampler in three dimensions:



Extended generator

State space $\mathbb{R}^d \times \{-1, +1\}^d$.

$$\mathcal{L}f(x, v) = v^\top \partial_x f(x, v) + \sum_{i=1}^{d} (v_i \partial_i U(x))^+ \, (\mathcal{F}_i - \mathrm{id}) f(x, v)$$

where $\mathcal{F}_i f(x, v) = f(x, F_i v)$ and F_i switches the i-th coordinate of v.

Motivation for using PDMPs

- Non-reversible $\rightsquigarrow$ Avoid diffusive behaviour (?)
- Scalable (I do not have enough time to explain this.)
- (Anyway,) Different from MCMC

Motivation for using PDMPs

- Non-reversible $\rightsquigarrow$ Avoid diffusive behaviour (?)
- Scalable (I do not have enough time to explain this.)
- (Anyway,) Different from MCMC

Packages

- **RZigZag** : Code: R, Algorithms: BPS, ZZ

Motivation for using PDMPs

- Non-reversible $\rightsquigarrow$ Avoid diffusive behaviour (?)
- Scalable (I do not have enough time to explain this.)
- (Anyway,) Different from MCMC

Packages

- **RZigZag** : Code: R, Algorithms: BPS, ZZ
- **PDSampler.jl** : Code: Julia, Algorithms: BPS

Motivation for using PDMPs

- Non-reversible $\rightsquigarrow$ Avoid diffusive behaviour (?)
- Scalable (I do not have enough time to explain this.)
- (Anyway,) Different from MCMC

Packages

- **RZigZag** : Code: R, Algorithms: BPS, ZZ
- **PDSampler.jl** : Code: Julia, Algorithms: BPS
- **ZigZagBoomerang** : Code: Julia, Algorithms: BPS, ZZ, Boomerang

Motivation for using PDMPs

- Non-reversible $\rightsquigarrow$ Avoid diffusive behaviour (?)
- Scalable (I do not have enough time to explain this.)
- (Anyway,) Different from MCMC

Packages

- **RZigZag** : Code: R, Algorithms: BPS, ZZ
- **PDSampler.jl** : Code: Julia, Algorithms: BPS
- **ZigZagBoomerang** : Code: Julia, Algorithms: BPS, ZZ, Boomerang
- Additionally, check out the brand new **pdmp-jax** by Charly Andral!

Summary

Summary

- The ergodicity of Markov chains is demonstrated through non-singularity and invariance.

Summary

- The ergodicity of Markov chains is demonstrated through non-singularity and invariance.
- **Markov Chain Monte Carlo (MCMC)** methods can approximate integrals, and we understand both how to design them and why they converge.

Summary

- The ergodicity of Markov chains is demonstrated through non-singularity and invariance.
- **Markov Chain Monte Carlo (MCMC)** methods can approximate integrals, and we understand both how to design them and why they converge.
- As an advanced topic, we treat Continuous-time methods, such as the **Zig-Zag Sampler** and **Bouncy Particle Sampler**.