UNIVERSITI MALAYSIA PAHANG

Chapter 6: Synthetic Data Generation

Noryanti Muhammad
Centre for Mathematical Sciences
College of Computing and Applied Sciences
Universiti Malaysia Pahang

Centre of Excellence (CoE) for Data Science & Artificial Intelligence
Research & Innovation Department
Universiti Malaysia Pahang

Expected Outcomes:

By the end of this chapter, students should be able:

- ✓ To understand the process creating the synthetics data set.
- ✓ To understand and use the bootstrapping method.
- ✓ To understand and use the Markov Chain Monte Carlo (MCMC) Method Including the Gibbs sampling and Metropolis-Hastings’s sampling method

Content:

6.1 Introduction

What is synthetic data?

- Synthetic data is data that you can create at any scale, whenever and wherever you need it.
- Crucially, synthetic data mirrors the balance and composition of real data, making it ideal for fueling machine learning models.
- What makes synthetic data special is that data scientists, developers and engineers are in complete control. There’s no need to put your faith in unreliable, incomplete data, or struggle to find enough data for machine learning at the scale you need. Just create it for yourself.

6.1 Introduction

Advantages of synthetic data

As a data scientists, the real or synthetic nature of data is irrelevant. What really matters are **the characteristics** and **patterns inside the data** – its **quality**, **balance** and **bias**.

Synthetic data allows you to **optimize** and **enrich** your data, unlocking several key benefits.

Increased data quality

Real-world data is not just hard and expensive to source. It is also prone to errors, inaccuracies and bias that can severely impact the quality of your machine learning model.

Increased confidence in data quality, variety and balance. From auto-completing missing values to automated labeling, it's a way to dramatically increase the reliability and accuracy of your data and, in turn, the accuracy of your predictions.

6.1 Introduction

Advantages of synthetic data

Scalability

Fueling the machine learning economy takes a huge amount of data. Few data scientists can access exactly the data they lack on the scale they need to test and train powerful predictive models. Synthetic data can close that gap.

Many data scientists supplement their real-world records with synthetic data, rapidly scaling up existing data – or just the relevant subsets of this data – to create more meaningful observations and trends

6.1 Introduction

Advantages of synthetic data

Powerful simplicity

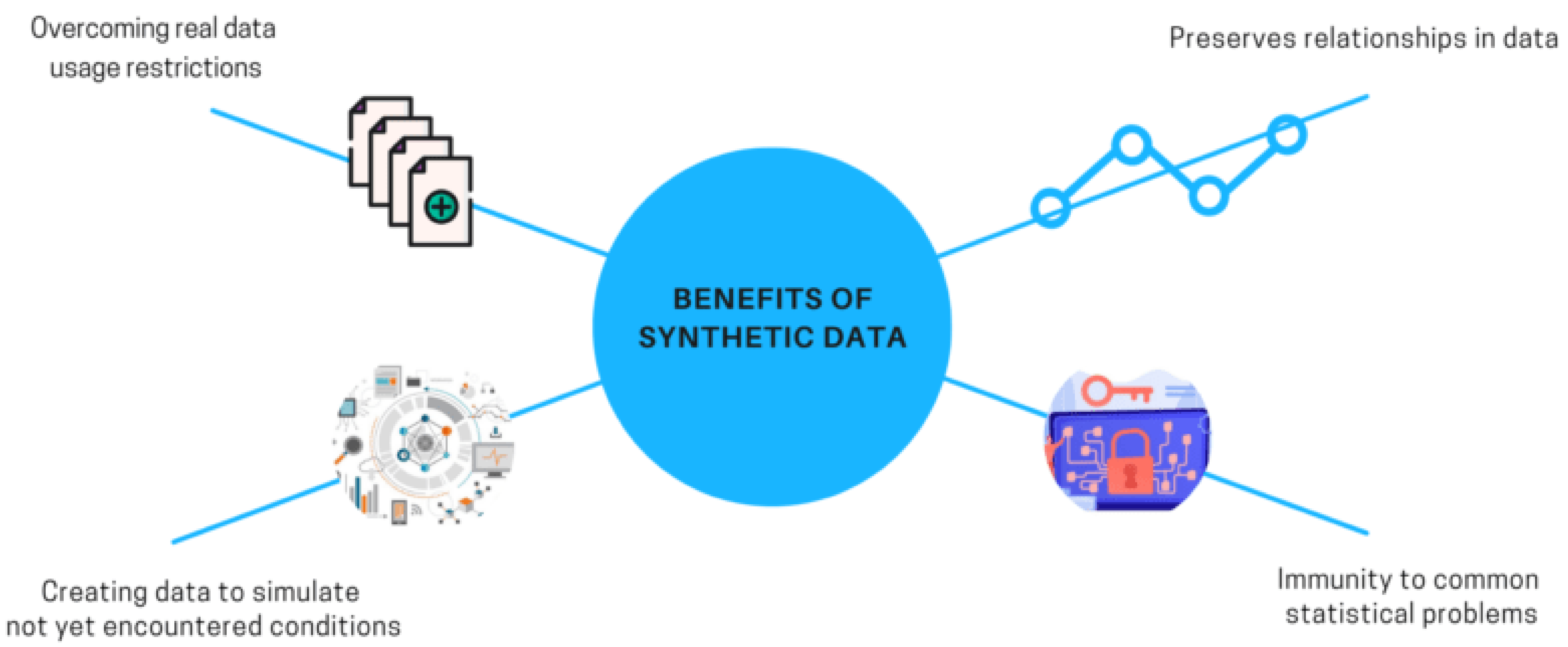
Synthetic data is refreshingly easy to generate. With real-world data, developers need to:

- Ensure privacy and confidentiality
- Label data in a uniform way
- Filter out duplicate data
- Remove erroneous records
- Collate data from multiple sources, often in multiple formats

Using the synthetic data, you can control how the resulting data is structured, formatted and labeled. That means a ready-to-use source of high-quality, dependable data is just a few clicks away.

6.1 Introduction

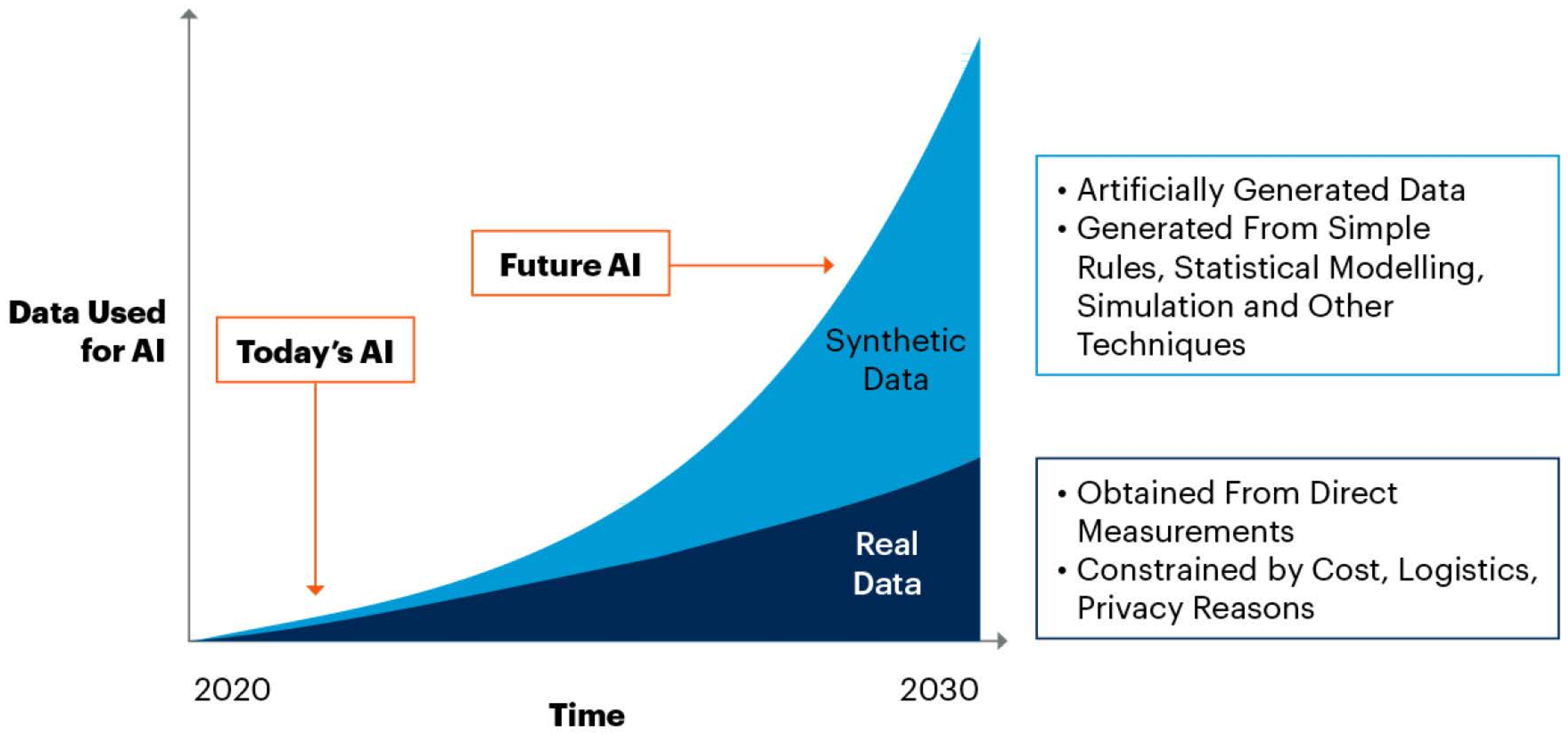
Advantages of synthetic data


Overcoming real data usage restrictions
Preserves relationships in data
BENEFITS OF SYNTHETIC DATA
Creating data to simulate not yet encountered conditions
Immunity to common statistical problems

6.1 Introduction

Future

By 2030, Synthetic Data Will Completely Overshadow Real Data in AI Models



- Artificially Generated Data
- Generated From Simple Rules, Statistical Modelling, Simulation and Other Techniques

- Obtained From Direct Measurements
- Constrained by Cost, Logistics, Privacy Reasons

Source: Gartner
750175_C

6.1 Introduction

Why important now?

Synthetic data is important because it can be generated to meet specific needs or conditions that are not available in existing (real) data. This can be useful in numerous cases such as

- When privacy requirements limit data availability or how it can be used.
- Data is needed for testing a product to be released however such data either does not exist or is not available to the testers
- Training data is needed for machine learning algorithms. However, especially in the case of self-driving cars, such data is expensive to generate in real life.

Though synthetic data first started to be used in the ’90s, an abundance of computing power and storage space of the 2010s brought more widespread use of synthetic data.

6.1 Introduction

Example of its application

Industries that can benefit from synthetic data:

- Automotive and Robotics
- Financial services
- Healthcare
- Manufacturing
- Security
- Social Media

Business functions that can benefit from synthetic data include:

- Marketing
- Machine learning
- Agile development and DevOps
- HR

Amazon Go uses synthetic data to train cashier-less store algorithms: https://venturebeat.com/2019/06/05/amazon-go-uses-synthetic-data-to-train-cashierless-store-algorithms/

Amazon drones and warehouse robots also use synthetic data to improve their efficiency and accuracy.

6.1 Introduction

Type of synthetic data (three types)

Fully synthetic

- This data **does not contain** any original data.
- This means that re-identification of any single unit is almost impossible, and all variables are still fully available.

Partially synthetic

- Only data that is sensitive is replaced with synthetic data.
- This requires a heavy dependency on the imputation model.
- This leads to decreased model dependence but does mean that some disclosure is possible owing to the true values that remain within the dataset.

6.1 Introduction

Type of synthetic data (three types)

Hybrid Synthetic Data

This data is generated using both real and synthetic data.

- For each random record of real data, a close record in the synthetic data is chosen and then both are combined to form hybrid data.
- It provides advantages of both fully and partially synthetic data.
- therefore, is known to provide good privacy preservation with high utility compared to the other two but at a fallback of more memory and processing time.

6.1 Introduction

Challenges of Synthetic Data

Outliers may be missing: Synthetic data can only mimic the real-world data; it is not an exact replica of it. Therefore, synthetic data may not cover some outliers that original data has. However, outliers in the data can be more important than regular data points as Nassim Nicholas Taleb explains in depth in his book, the Black Swan.

Quality of the model depends on the data source: The quality of synthetic data is highly correlated with the quality of the input data and the data generation model. Synthetic data may reflect the biases in source data

User acceptance is more challenging: Synthetic data is an emerging concept, and it may not be accepted as valid by users who have not witnessed its benefits before.

Synthetic data generation requires time and effort: Though easier to create than actual data, synthetic data is also not free.

Output control is necessary: Especially in complex datasets, the best way to ensure the output is accurate is by comparing synthetic data with authentic data or human-annotated data. this is because there could be inconsistencies in synthetic data when trying to replicate complexities within original datasets

6.1 Introduction

Methodologies: Majorly there are two ways to generate synthetic data

1. **Drawing numbers from a distribution:**

The key idea is **to observe the statistical distribution of real-world data** and **then replicate the same to produce similar data** with simple numbers.

2. **Agent-based modeling:**

The key idea is **to create a physical model of the observed statistical distribution of real-world data**, then **reproduce random data using the same model.** It focuses on understanding the impact of the interaction between agents that directly affects the system as a whole.

6.3 Bootstrapping method

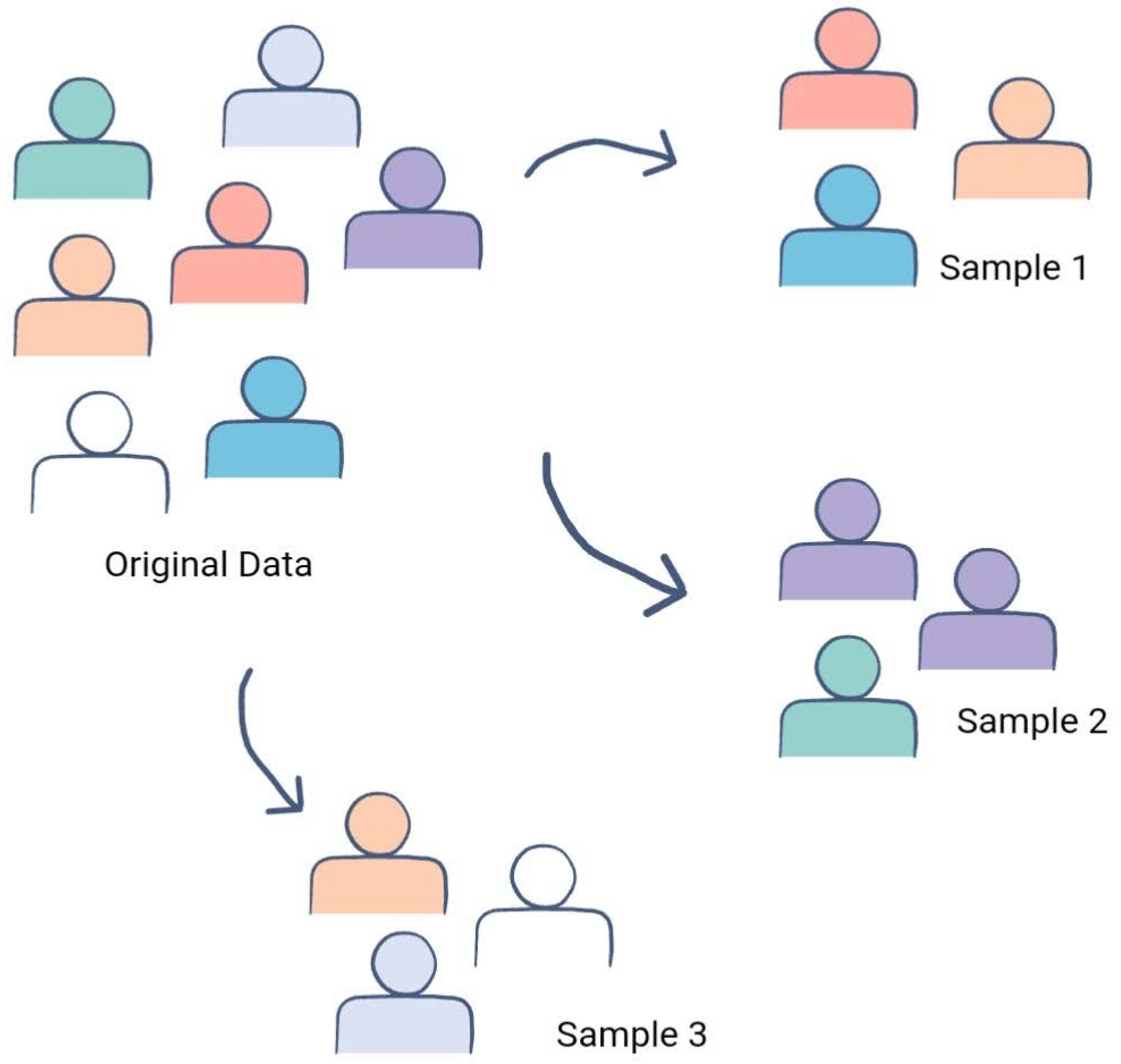
Introduction

- Bootstrapping is a resampling method used to stimulate samples out of a data set using the replacement technique.
- The process of bootstrapping allows one to infer data about the population, derive standard errors, and ensure that data is tested efficiently.
- Bootstrapping is a resampling statistical technique that evaluates statistics of a given population by testing a dataset by replacing the sample.
- This technique involves repeatedly sampling a dataset with random replacement. A statistical test that falls under the category of resampling methods, this method ensures that the statistics evaluated are accurate and unbiased as much as possible.

6.3 Bootstrapping method

Introduction

- The Bootstrapping method uses the samples procured from a study over and over again in order to use the replacement technique and ensure that the stimulated samples lead to an accurate evaluation.
- Other than ensuring the sampling of the accuracy of a given dataset, bootstrapping in statistics also allows one to estimate the confidence intervals of a given dataset.



6.3 Bootstrapping method

The procedure

Bootstrapping **resamples the original dataset** with **replacement** many thousands of times to create simulated datasets. This process involves drawing random samples from the original dataset.

- The bootstrap method has an **equal probability** of randomly **drawing each original** data point for inclusion in the resampled datasets.
- The procedure can **select a data point more than once** for a resampled dataset. This property is the "with replacement" aspect of the process.
- The procedure creates **resampled datasets that are the same size** as the original dataset.

The process ends with your simulated datasets **having many different combinations of the values** that exist in the original dataset.

Each simulated dataset has **its own set of sample statistics**, such as the mean, median, and standard deviation.

Bootstrapping procedures **use the distribution of the sample statistics** across the simulated samples as the sampling distribution.

6.3 Bootstrapping method

The procedure

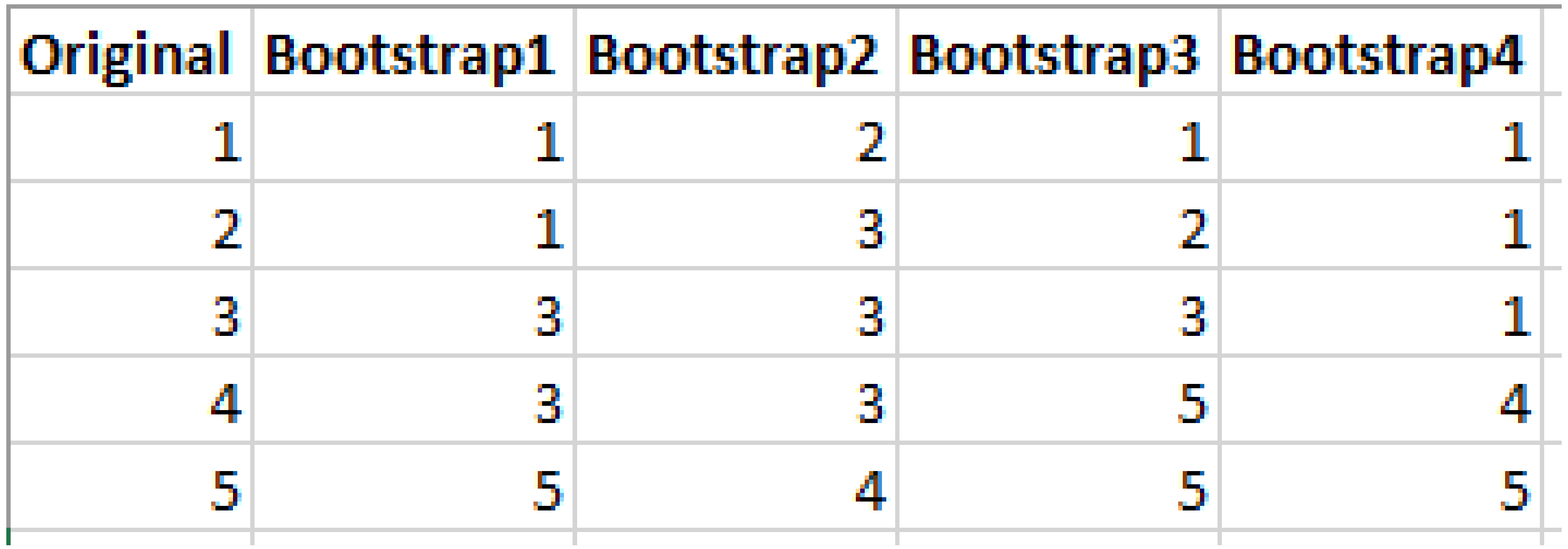
Generally, the steps to create a bootstrap sample:

1. Replace the population with the sample
2. Sample with replacement K times. K should be large, say 1000.
3. Compute sample medians each time, Mi
4. Obtain the approximate distribution of the sample median.

6.3 Bootstrapping method

Simple Example

Suppose a study collects five data points and creates four bootstrap samples,



Original	Bootstrap1	Bootstrap2	Bootstrap3	Bootstrap4
1	1	2	1	1
2	1	3	2	1
3	3	3	3	1
4	3	3	5	4
5	5	4	5	5

6.3 Bootstrapping method

Type of Bootstrap

Parametric Bootstrap Method

- In this method, the distribution parameter must be known.
- This means that the assumption of the kind of distribution the sample has must be provided beforehand.
- For instance, it must be known to the user if the sample has Gaussian Distribution or Skewed distribution.
- This type of bootstrap method is more efficient since it already knows the nature of distribution.

Non-Parametric Bootstrap Method

- Unlike the parametric bootstrap method, this type does not require the parameter of distribution to be known beforehand.
- Therefore, this type of bootstrap method works without assuming the nature of the sample distribution.

6.3 Bootstrapping method

Type of Bootstrap

- Case resampling
- Estimating the distribution of sample mean
- Bayesian bootstrap
- Smooth bootstrap
- Parametric bootstrap
- Resampling residuals
- Gaussian process regression bootstrap
- Wild bootstrap
- Block bootstrap (Time Series)

Regression

- In regression problems, case resampling refers to the simple scheme of resampling individual cases – often rows of a data set.
- For regression problems, if the data set is large, this simple scheme is often acceptable. However, the method is open to criticism
- In regression problems, the explanatory variables are often fixed, or at least observed with more control than the response variable.
- Also, the range of the explanatory variables defines the information available from them.
- Therefore, to resample cases means that each bootstrap sample will lose some information. As such, alternative bootstrap procedures should be considered..

6.3 Bootstrapping method

Simple Example

```
sample(x, size, replace, prob)

Refer Bootstrap0
```

X is a vector containing the data set to be resampled or the indices of the data to be resampled

The **size** option specifies the sample size with the default being the size of the population being resampled

he **replace** option determines if the sample will be drawn with or without replacement where the default value is FALSE, i.e. without replacement

The **prob** option takes a vector of length equal to the data set given in the first argument containing the probability of selection for each element of x.

6.3 Bootstrapping method

Simple Example

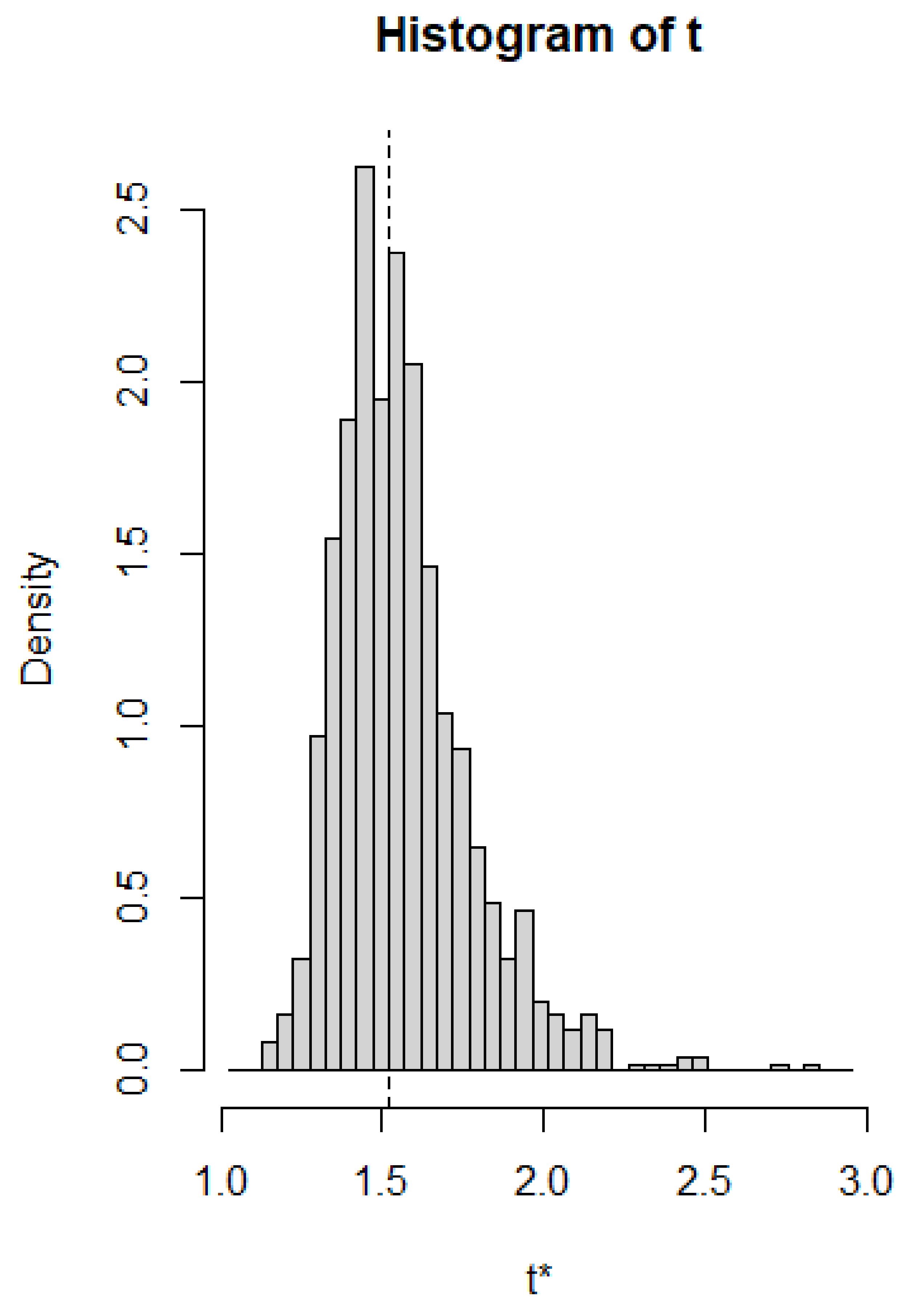
Consider to obtain a standard error for the estimate of the median.

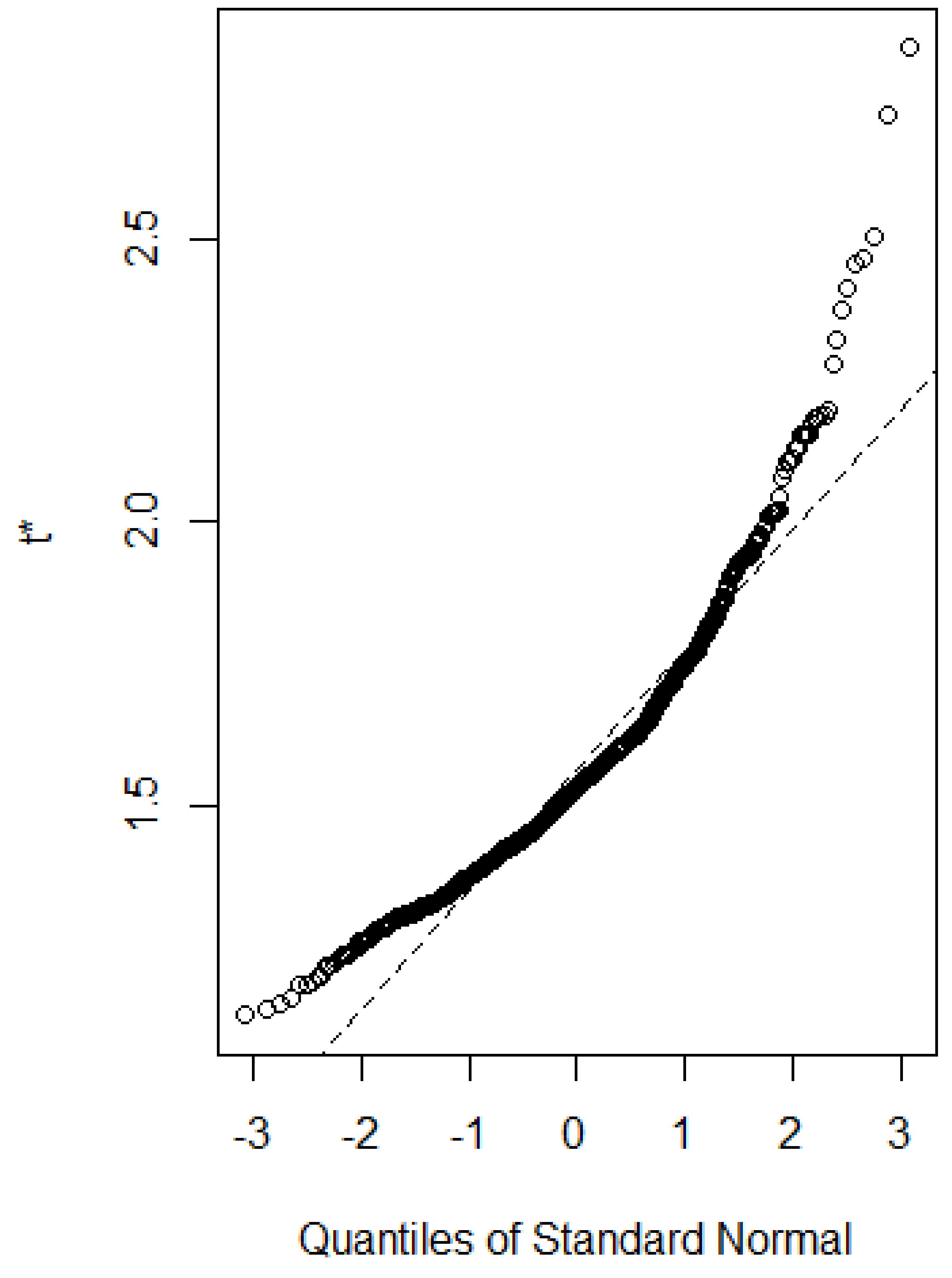
Can be done using the lapply, sapply functions in combination with the sample function.

```
#function which will bootstrap the standard error of
the median
b.median <- function(data, num) {
    resamples <- lapply(1:num, function(i) sample(data,
replace=T))
    r.median <- sapply(resamples, median)
    std.err <- sqrt(var(r.median))
    list(std.err=std.err, resamples=resamples,
medians=r.median)
}
#generating the data to be used (same as in the above
example)
data1 <- round(rnorm(100, 5, 3))

#saving the results of the function b.median in the
object b1
b1 <- b.median(data1, 30)

#displaying the first of the 30 bootstrap samples
b1$resamples[1]
```

6.3 Bootstrapping method

Example of Bootstrapping: R package boot





```
Refer Codding Bootstrap2

# Usual bootstrap of the ratio of means using
the city data
ratio <- function(d, w) sum(d$x * w)/sum(d$u
* w)
output <-boot(city, ratio, R = 999, stype =
"w")

# Plotting the output
output
plot(output)
# Obtaining a confidence interval of 95%
boot.ci(output, type="bca")
```

TEKNOLOGI UNTUK MASYARAKAT

UMPMalaysia

6.3 Bootstrapping method

Example of Bootstrapping: R package boot

Refer Codding Bootstrap1

Consider to perform bootstrapping on a single statistic (k = 1). And, make use of the dataset – ‘`mtcars`’.

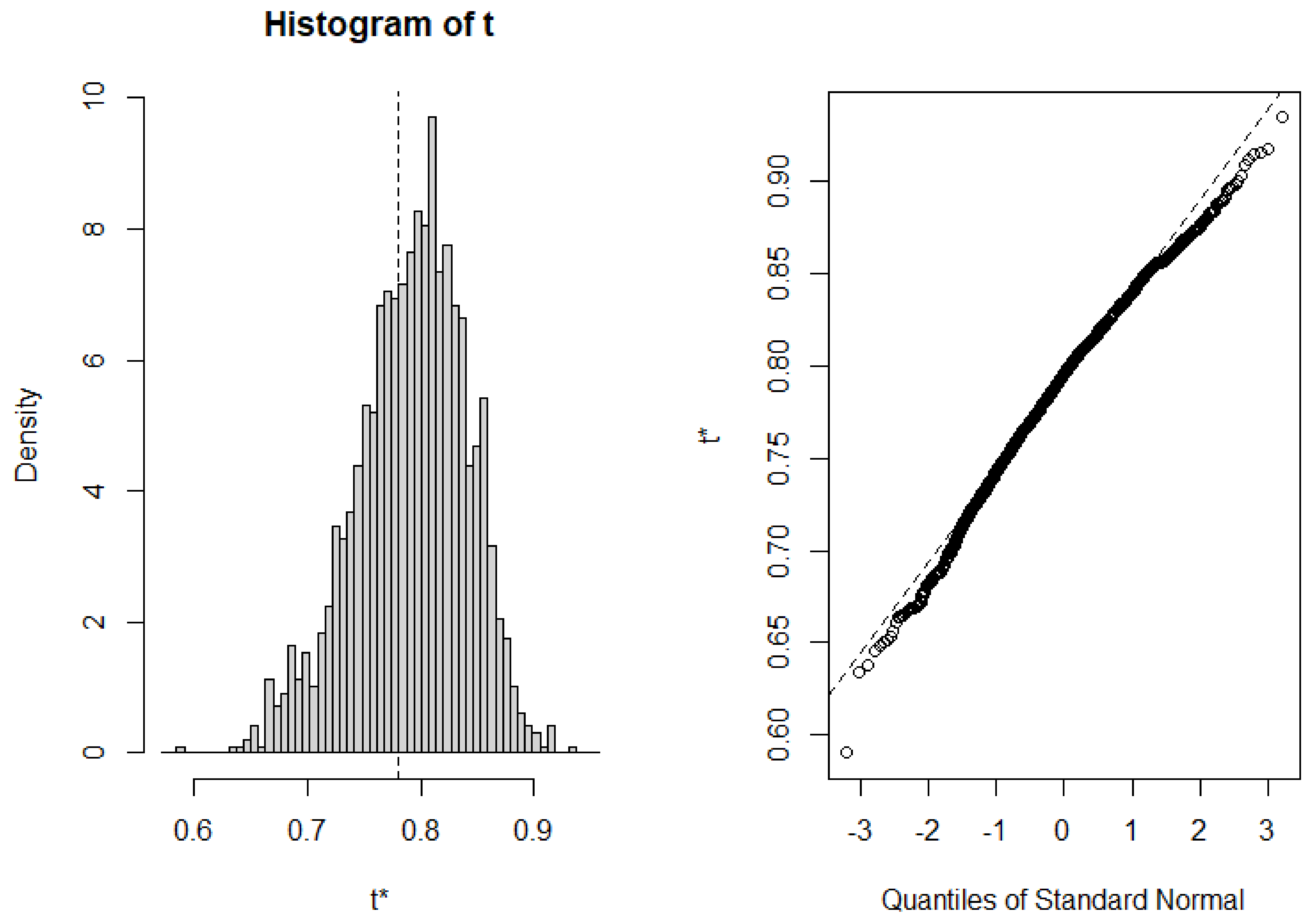
We will obtain a bootstrapped confidence interval of 95% for the R-squared in the linear regression relationship of miles per gallon variable (mpg) on car weight (wt) and its displacement (disp).

This bootstrapped confidence interval is based on 1500 replications.

```
# Package
library(boot)
# Creating Function to obtain R-Squared from
the data
r_squared <- function(formula, data, indices)
{
val <- data[indices,] # selecting sample with
boot
fit <- lm(formula, data=val)
return(summary(fit)$r.square)
}
# Performing 1500 replications with boot
output <- boot(data=mtcars,
statistic=r_squared,
R=1500, formula=mpg~wt+disp)
# Plotting the output
output
plot(output)
# Obtaining a confidence interval of 95%
boot.ci(output, type="bca")
```

6.3 Bootstrapping method

Example of Bootstrapping: R package boot



6.3 Bootstrapping method

Example of Bootstrapping: R package boot

Refer Codding Bootstrap3

- In this example we show the use of boot in a prediction from regression based on the nuclear data. This example is taken from Example 6.8 of Davison and Hinkley (1997). Notice also that two extra arguments to 'statistic' are passed through boot.

- We set up a new data frame with the data, the standardized residuals and the fitted values for use in the bootstrap.

- Now we want a prediction of plant number 32 but at date 73.00 (prediction need m value)

- Then, obtaining a confidence interval of 95% and find the bootstrap prediction squared error

6.3 Bootstrapping method

Example of Bootstrapping: R package boot

Refer Codding Bootstrap4

Finally, a parametric bootstrap.

For this example, consider at the air-conditioning data.

In this example the aim is to test the hypothesis that the true value of the index is 1 (i.e. that the data come from an exponential distribution) against the alternative that the data come from a gamma distribution with index not equal to 1.

6.4 Markov Chain Monte Carlo Methods

Monte Carlo Simulation

- Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables.

- It is a technique used to understand the impact of risk and uncertainty in prediction and forecasting models.

- A Monte Carlo simulation can be used to tackle a range of problems in virtually every field such as finance, engineering, supply chain, and science. It is also referred to as a multiple probability simulation.

6.4 Markov Chain Monte Carlo Methods

Understanding Monte Carlo Simulations

- When faced with significant uncertainty in the process of making a forecast or estimation, rather than just replacing the uncertain variable with a single average number, the Monte Carlo Simulation might prove to be a better solution by using multiple values

- **Example:**
  - They are used to estimate the probability of cost overruns in large projects and the likelihood that an asset price will move in a certain way.
  - Telecoms use them to assess network performance in different scenarios, helping them to optimize the network.
  - Analysts use them to assess the risk that an entity will default, and to analyze derivatives such as options.

6.4 Markov Chain Monte Carlo Methods

Monte Carlo Simulation History

- Monte Carlo simulations are named after the popular gambling destination in **Monaco**, since chance and random outcomes are central to the modeling technique, much as they are to games like roulette, dice, and slot machines.
- The technique was first developed by Stanislaw Ulam, a mathematician who worked on the Manhattan Project.
- After the war, while recovering from brain surgery, Ulam entertained himself by playing countless games of solitaire.
- He became interested in plotting the outcome of each of these games in order to observe their distribution and determine the probability of winning.
- After he shared his idea with John Von Neumann, the two collaborated to develop the Monte Carlo simulation.

6.4 Markov Chain Monte Carlo Methods

Monte Carlo Simulation Method

- The basis of a Monte Carlo simulation is that the probability of varying outcomes cannot be determined because of random variable interference.
- Therefore, a Monte Carlo simulation focuses on constantly repeating random samples to achieve certain results.

1. A Monte Carlo simulation takes the variable that has uncertainty and assigns it a random value.
2. The model is then run, and a result is provided.
3. This process is repeated again and again while assigning the variable in question with many different values.
4. Once the simulation is complete, the results are averaged together to provide an estimate.

6.4 Markov Chain Monte Carlo Methods

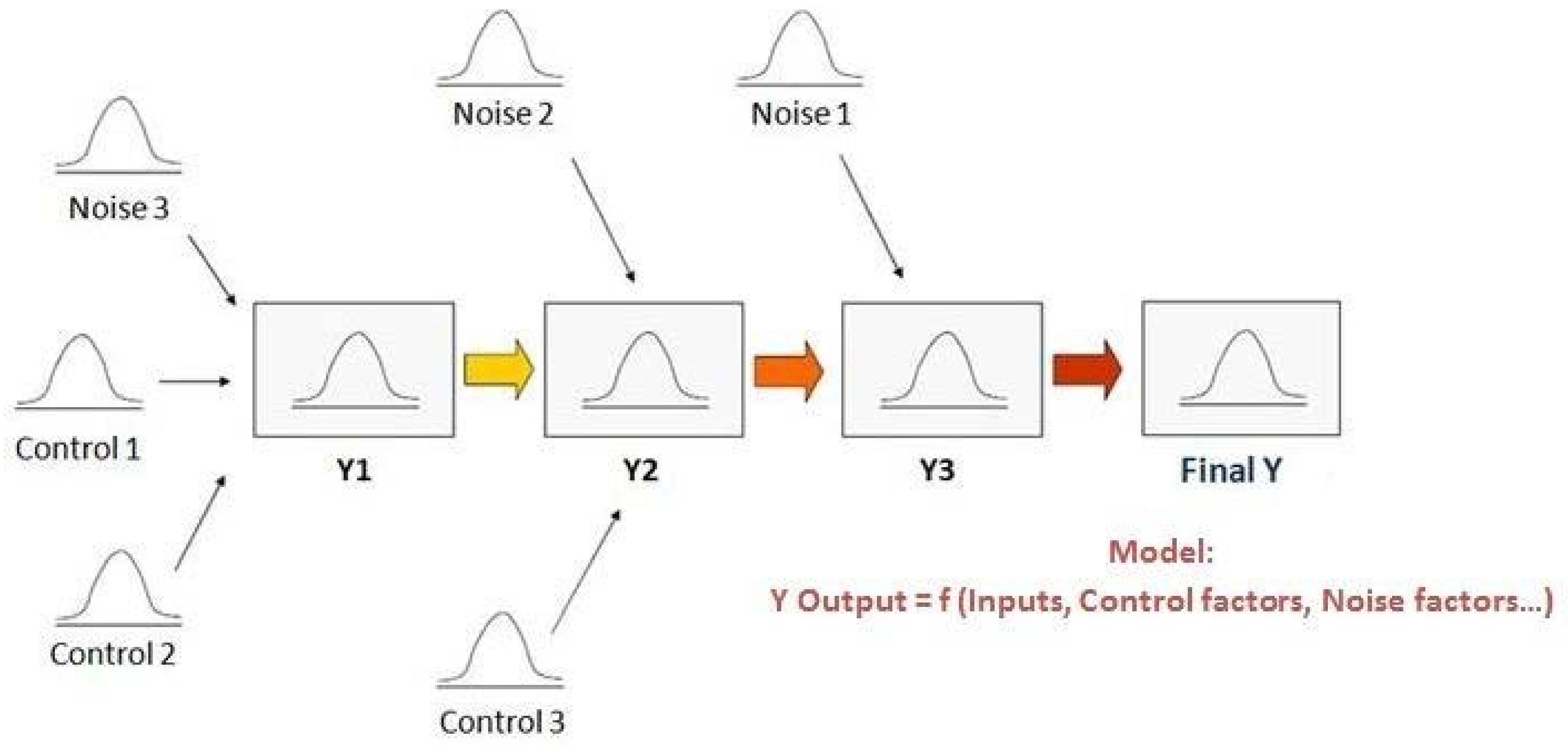
Monte Carlo Simulation Method

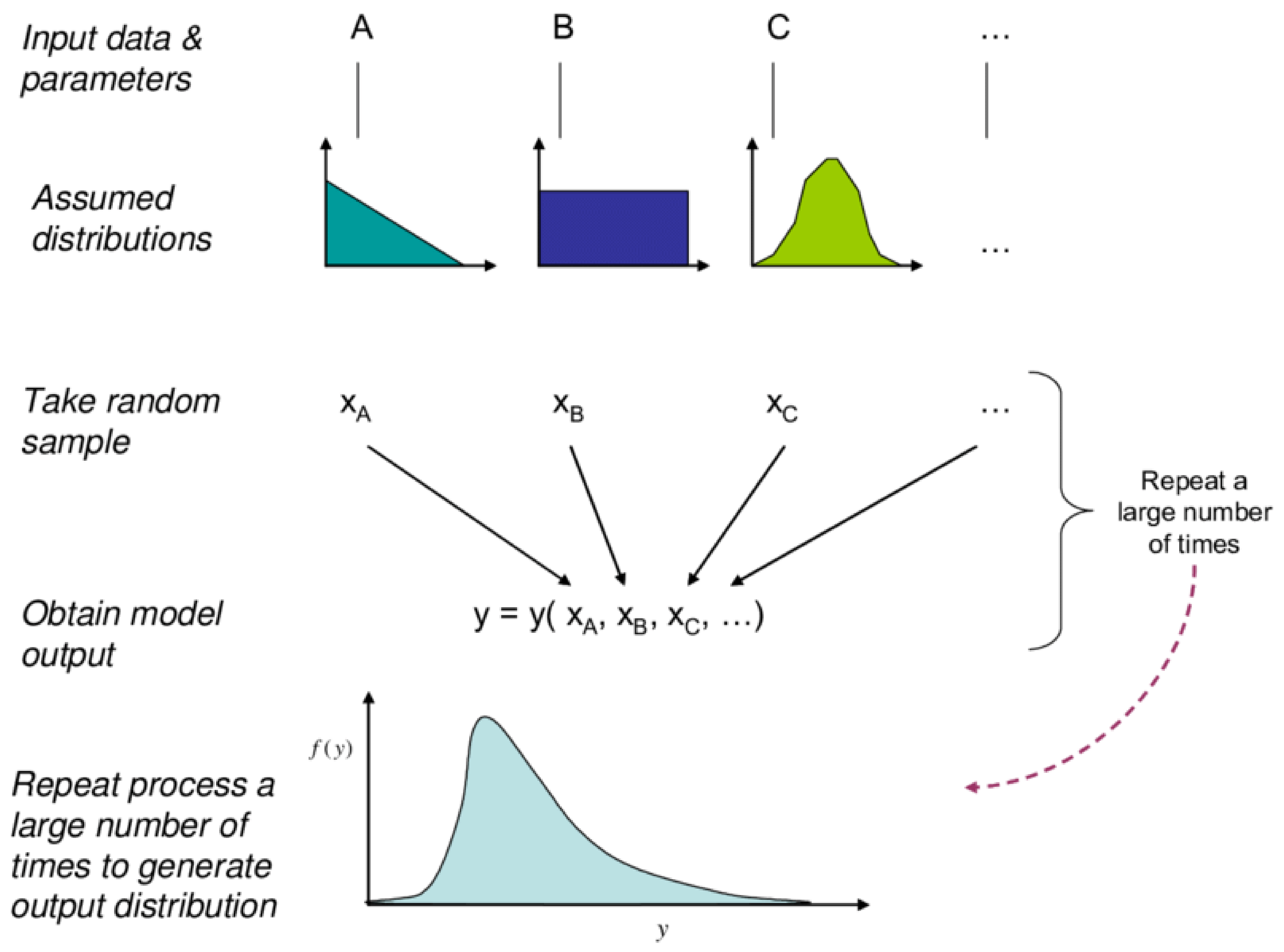
- The basis of a Monte Carlo simulation is that the probability of varying outcomes cannot be determined because of random variable interference.
- Therefore, a Monte Carlo simulation focuses on constantly repeating random samples to achieve certain results.

1. A Monte Carlo simulation takes the variable that has uncertainty and assigns it a random value.
2. The model is then run, and a result is provided.
3. This process is repeated again and again while assigning the variable in question with many different values.
4. Once the simulation is complete, the results are averaged together to provide an estimate.

6.4 Markov Chain Monte Carlo Methods

Monte Carlo Simulation Method




Input data & parameters
A
B
C
...
Assumed distributions
Take random sample
x_A
x_B
x_C
Obtain model output
$y = y(x_A, x_B, x_C, \dots)$
Repeat a large number of times
Repeat process a large number of times to generate output distribution
$f(y)$
y

6.4 Markov Chain Monte Carlo Methods

Monte Carlo Simulation Method in R

- To use Monte Carlo methods, we need to be able **to replicate some random process many times.**

- There are two main ways this is commonly done: either with `replicate()` or with `for()` loops.

Consider that we have a vector **x**, which represents 30 observations of fish length (mm).

We wish to build the sampling distribution of the mean length “by hand”. We can sample randomly from it, calculate the mean, then repeat this process many times:

```
x = rnorm(30, 500, 30)
means = replicate(n = 1000, expr = {
  x_i = sample(x, length(x), replace = T)
  mean(x_i)
})
```

6.4 Markov Chain Monte Carlo Methods

EXAMPLE

Build a (very) basic population model.

At the start of the first year, **the population abundance** is 1000 individuals and grows by an average factor of 1.1 per year (reproduction and death processes result in a growth rate of 10%) before harvest.

The growth rate varies randomly, however. Each year, the 1.1 growth factor has variability introduced by small changes in survival and reproductive process.

Model these variations as lognormal random variables.

After production, 8% of the population is harvested. Simulate the abundance at the end of the year for 100 years

```
nt = 100          # number of years
N = NULL          # container for
abundance
N[1] = 1000       # first end-of-year
abundance

for (t in 2:nt) {
  # N this year is N last year * growth
*
    # randomness * fraction that
survive harvest
  N[t] = (N[t-1] * 1.1 * rlnorm(1, 0,
0.1)) * (1 - 0.08)
}

plot(N, type = "l", pch = 15, xlab =
"Year", ylab = "Abundance")
```

6.4 Markov Chain Monte Carlo Methods

EXAMPLE

In Monte Carlo analyses, it is often useful to **wrap code into functions**.

This allows **for easy replication** and **setting adjustment** (e.g., if you wanted to compare the growth trajectories of two populations with differing growth rates).

As an example, turn the population model shown into a function:

The function takes five inputs:

nt: the number of years,
grow: the population growth rate,
sd_grow: the amount of annual variability in the growth rate
U: the annual exploitation rate
plot: whether you wish to have a plot created. It has a default setting of FALSE: if you don't specify plot = T when you call pop_sim(), you won't see a plot made.
It returns one output: the vector of population abundance.

```
pop_sim = function(nt, grow, sd_grow,
U, plot = F) {
  N = NULL # empty flexible vector
container
  N[1] = 1000
  for (t in 2:nt) {
    N[t] = (N[t-1] * grow * rlnorm(1,
0, sd_grow)) * (1 - U)
  }

  if (plot) {
    plot(N, type = "l", pch = 15, xlab
= "Year", ylab = "Abundance")
  }

  N
}
#Execute
pop_sim(100, 1.1, 0.1, 0.08, T)
```

6.4 Markov Chain Monte Carlo Methods

EXAMPLE

Now, you wish to replicates 1000 times, use `replicate()`function.

```
out = replicate(n = 1000, expr = pop_sim(100, 1.1, 0.1, 0.08, F))
```

The function takes five inputs:

nt: the number of years,
grow: the population growth rate,
sd_grow: the amount of annual variability in the growth rate
U: the annual exploitation rate
plot: whether you wish to have a plot created. It has a default setting of FALSE: if you don't specify plot = T when you call pop_sim(), you won't see a plot made.
It returns one output: the vector of population abundance.

6.4 Markov Chain Monte Carlo Methods

EXAMPLE: Conduct a power analysis using stochastic simulation (i.e., a Monte Carlo analysis)

A power analysis is one where the analyst wishes to determine how much power they will have to detect an effect.

Power is inversely related to the probability of making a Type II Error: failing to reject a false null hypothesis. In other words, having high power means that you have a high chance of detecting an effect if an effect truly exists.

Power is a function of the effect size, the sample size *n*, and the variability in the data.

Strong effects are easier to detect than weak ones, more samples increase the test’s sensitivity (the ability to detect weak effects), and lower variability results in more power.

6.4 Markov Chain Monte Carlo Methods

EXAMPLE: Conduct a power analysis using stochastic simulation (i.e., a Monte Carlo analysis)

Write R code of a power analysis to determine how likely are you to be able to correctly identify what you deem to be a biologically-meaningful difference in survival between two tagging procedures.

You know one tagging procedure has approximately a 10% mortality rate (10% of tagged fish die within the first 12 hours as result of the tagging process).

Another cheaper, and less labor-intensive method has been proposed, but before implementing it, your agency wishes to determine if it will have a meaningful impact on the reliability of the study or on the ability of the crew to tag enough individuals that will survive long enough to be useful.

6.4 Markov Chain Monte Carlo Methods

EXAMPLE: Conduct a power analysis using stochastic simulation (i.e., a Monte Carlo analysis)

You and your colleagues determine that if the mortality rate of the new tagging method reaches 25%, then gains in time and cost-efficiency would be offset by needing to tag more fish (because more will die).

You have decided to perform a small-scale study to determine if using the new method could result in 25% or more mortality. The study will tag *n* individuals using both methods (new and old) and track the fraction that survived after 12 hours.

Before performing the study however, you deem it important to determine how large *n* needs to be to answer this question.

You decide to use a stochastic power analysis to help your research group.

The small-scale study can tag a total of at most 100 fish with the currently available resources.

Could you tag fewer than 100 total individuals and still have a high probability of detecting a statistically significant difference in mortality?

6.4 Markov Chain Monte Carlo Methods

EXAMPLE: Conduct a power analysis using stochastic simulation (i.e., a Monte Carlo analysis)

Step 1: The stochastic power analysis approach works like this (this is called psuedocode):

1. Simulate data under the reality that the difference is real with *n* observations per treatment, where $n < 100/2$.
2. Fit the model that will be used when the real data are collected to the simulated data.
3. Determine if the difference was detected with a significant *p*-value.
4. Replicate steps 1 - 3 many times.
5. Replicate step 4 while varying n over the interval from 10 to 50.
6. Determine what fraction of the p-values were deemed significant at each *n*.

Refer R Code MC2

Step 2 Require fitting a **generalized linear model**; for a review, revisit Chapter 3 (specifically on logistic regression).

6.4 Markov Chain Monte Carlo Methods

EXAMPLE: Conduct a power analysis using stochastic simulation (i.e., a Monte Carlo analysis)

Suppose you and your colleagues are not relying on p-values in this case, and are purely interested in how precisely the effect size would be estimated.

Adapt your function to determine how frequently you would be able to estimate the true mortality of the new method within +/- 5% based on the point estimate only (the estimate for the tagging mortality of the new method must be between 0.2 and 0.3 for a successful study).

Change your function to calculate this additional metric and re-run the analysis:

Refer R Code MC3

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC)

- One particularly popular subset of Monte Carlo methods is known as **Markov Chain Monte Carlo (MCMC)**.
- MCMC methods are appealing because they provide a straightforward, intuitive way to both simulate values from an unknown distribution and use those simulated values to perform subsequent analyses. Therefore, MCMC applicable in a wide variety of domains.
- Markov Chain Monte Carlo sampling provides a class of algorithms for systematic random sampling from high-dimensional probability distributions.

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC)

- Markov Chain Monte Carlo methods draw samples where the next sample is dependent on the existing sample, called a Markov Chain.
- This allows the algorithms to narrow in on the quantity that is being approximated from the distribution, even with a large number of random variables.
- By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain.
- The more steps are included, the more closely the distribution of the sample matches the actual desired distribution.
- Various algorithms exist for constructing chains, including the Metropolis–Hastings algorithm.

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC)

- Definition **- Markov chain** is a systematic method for generating a sequence of random variables where the current value is probabilistically dependent on the value of the prior variable. Specifically, selecting the next variable is only dependent upon the last variable in the chain.

- **Example**: Consider a board game that involves rolling dice, such as snakes and ladders (or chutes and ladders). The roll of a die has a uniform probability distribution across 6 stages (integers 1 to 6). You have a position on the board, but your next position on the board is only based on the current position and the random roll of the dice. Your **specific positions on the board** form a **Markov chain**.

6.4 Markov Chain Monte Carlo Methods

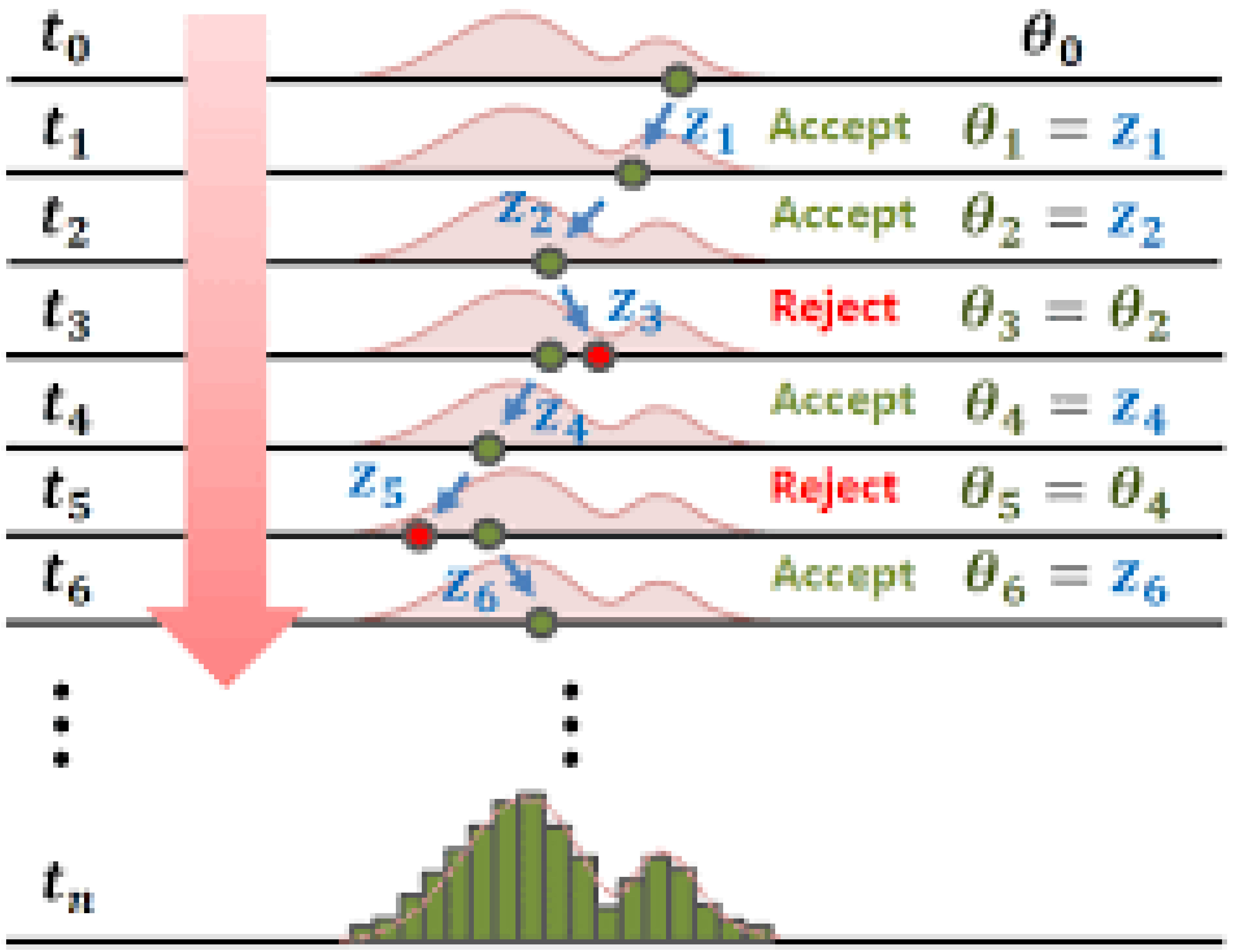
Markov Chain Monte Carlo (MCMC)

- **Example**: Another example of a Markov chain is a random walk in one dimension, where the possible moves are 1, -1, chosen with equal probability, and the next point on the number line in the walk is only dependent upon the current position and the randomly chosen move.

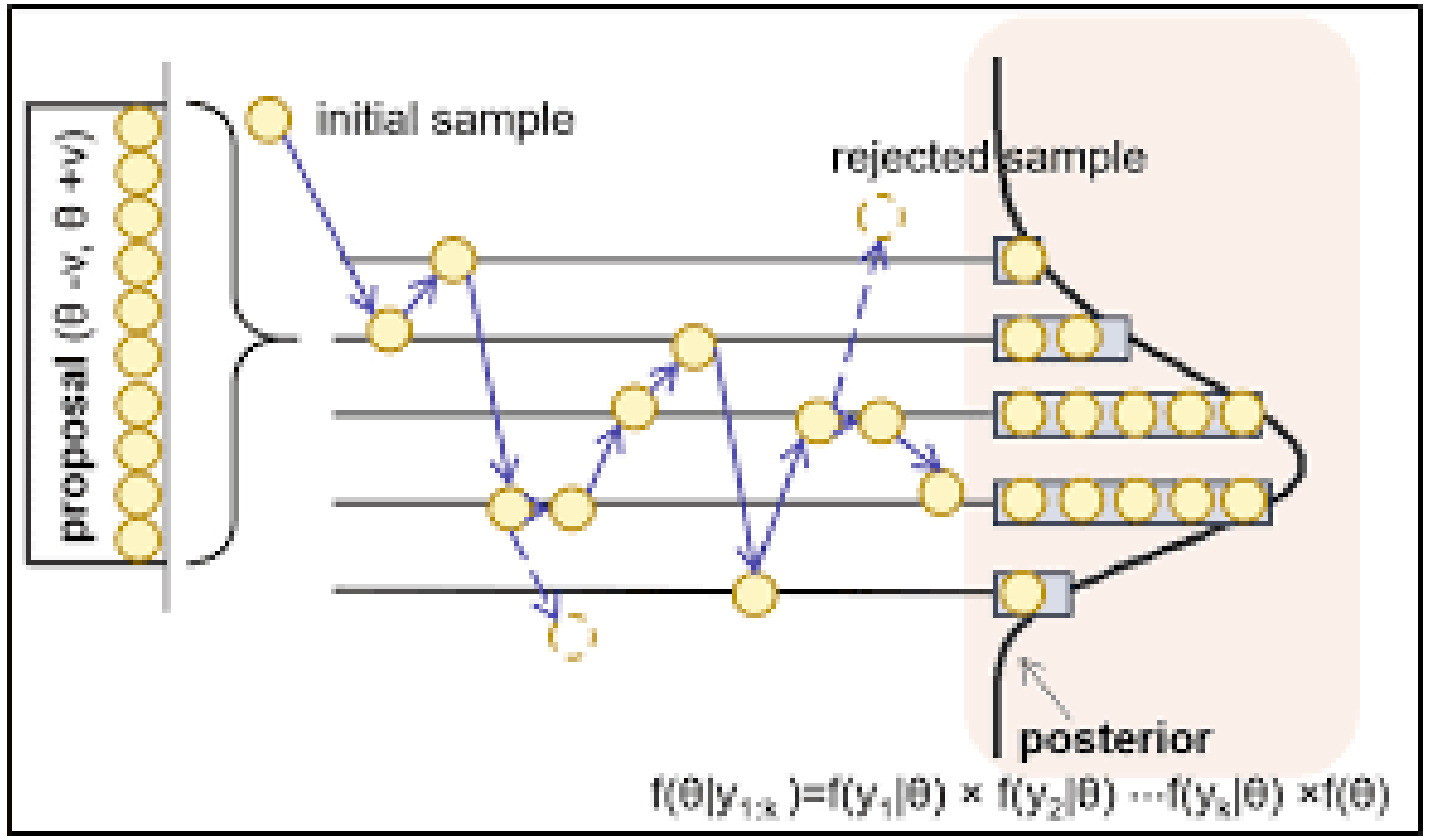
- At a high level, a Markov chain is defined in terms of a graph of states over which the sampling algorithm takes a random walk.

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC)



t_0 θ_0
t_1 z_1 Accept $\theta_1 = z_1$
t_2 z_2 Accept $\theta_2 = z_2$
t_3 z_3 Reject $\theta_3 = \theta_2$
t_4 z_4 Accept $\theta_4 = z_4$
t_5 z_5 Reject $\theta_5 = \theta_4$
t_6 z_6 Accept $\theta_6 = z_6$
t_n



proposal (θ -v, θ +v)
initial sample
rejected sample
posterior
$f(\theta|y_{1:k}) = f(y_1|\theta) \times f(y_2|\theta) \cdots f(y_k|\theta) \times f(\theta)$

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC)

- Specifically, MCMC is for performing inference (e.g. estimating a quantity or a density) for probability distributions where independent samples from the distribution cannot be drawn, or cannot be drawn easily.
- Samples are drawn from the probability distribution by constructing a Markov Chain, where the next sample that is drawn from the probability distribution is dependent upon the last sample that was drawn. The idea is that the chain will settle on (find equilibrium) on the desired quantity we are inferring.

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo Algorithms

- There are many Markov Chain Monte Carlo algorithms that mostly define different ways of constructing the Markov Chain when performing each Monte Carlo sample.
- The random walk provides a good metaphor for the construction of the Markov chain of samples, yet it is very inefficient. Consider the case where we may want to calculate the expected probability; it is more efficient to zoom in on that quantity or density, rather than wander around the domain.
- Markov Chain Monte Carlo algorithms are attempts at carefully harnessing properties of the problem in order to construct the chain efficiently.

6.4 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo Algorithms

- MCMC algorithms are sensitive to their starting point, and often require a warm-up phase or burn-in phase to move in towards a fruitful part of the search space, after which prior samples can be discarded and useful samples can be collected.
- Additionally, it can be challenging to know whether a chain has converged and collected a sufficient number of steps. Often a very large number of samples are required and a run is stopped given a fixed number of steps.
- The most common general Markov Chain Monte Carlo algorithm is called **Gibbs Sampling**; a more general version of this sampler is called the **Metropolis-Hastings algorithm.**

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm

- The Gibbs Sampling algorithm is an approach to constructing a Markov chain where the probability of the next sample is calculated as the conditional probability given the prior sample.

- Samples are constructed by changing one random variable at a time, meaning that subsequent samples are very close in the search space, e.g. local. As such, there is some risk of the chain getting stuck.

The idea behind Gibbs sampling is that we sample each variable in turn, conditioned on the values of all the other variables in the distribution.

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm

- Gibbs Sampling is appropriate for those probabilistic models where this conditional probability can be calculated, e.g. the distribution is discrete rather than continuous.

- Although this sampling step is easy for discrete graphical models, in continuous models, the conditional distribution may not be one that has a parametric form that allows sampling, so that Gibbs is not applicable.

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm

- Suppose $p(x, y)$ is a p.d.f. or p.m.f. that is difficult to sample from directly.
- Suppose, though, that we can easily sample from the conditional distributions $p(x|y)$ and $p(y|x)$.
- The Gibbs sampler proceeds as follows:

1. Set x and y to some initial starting values
2. Then sample **x|y**, then sample **y|x**,
3. Then **x|y**, and so on.

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm

STEP 0

0. Set (x0, y0) to some starting value.

STEP 1

1. Sample x1 ∼ p(x|y0), that is, from the conditional distribution X | Y = y0.

Current state: (x1, y0)

Sample y1 ∼ p(y|x1), that is, from the conditional distribution Y | X = x1.

Current state: (x1, y1)

STEP 2

2. Sample x2 ∼ p(x|y1), that is, from the conditional distribution X | Y = y1.

Current state: (x2, y1)

Sample y2 ∼ p(y|x2), that is, from the conditional distribution Y | X = x2.

Current state: (x2, y2)

Repeat iteration Step 1 and Step 2, M times

This procedure defines a sequence of pairs of random variables (X0, Y0),(X1, Y1),(X2, Y2),(X3, Y3), . . .

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm – Markov Chain and dependence

$(X_0, Y_0),(X_1, Y_1),(X_2, Y_2),(X_3, Y_3), \dots$ satisfies the property of being a Markov chain.

The conditional distribution of (X_i , Y_i) given all of the previous pairs depends only on (X_{i-1}, Y_{i-1})

$(X_0, Y_0),(X_1, Y_1),(X_2, Y_2),(X_3, Y_3), \dots$ are **not iid** samples (**Think about why**).

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm – Ideal Properties of MCMC

- (x_0, y_0) chosen to be in a region of high probability under $p(x, y)$, but often this is not so easy.
- We run the chain for M iterations and discard the first B samples $(X_1, Y_1), . . . ,(X_B, Y_B)$. This is called burn-in.
- Typically: if you run the chain long enough, the choice of B does not matter.
- Roughly speaking, the performance of an MCMC algorithm—that is, how quickly the sample averages $\frac{1}{N}\sum_{i=1}^{N} h(x_i, y_i)$ converge—is referred to as the mixing rate.
- An algorithm with good performance is said to “have good mixing”, or “mix well”.

6.4 Markov Chain Monte Carlo Methods

Exponential Example: Gibbs Sampling Algorithm

Consider the following Exponential model for observation(s), $x = (x_1, \ldots, x_n)^1$*:*

$$p(x|a, b) = ab\exp(-abx)I(x > 0)$$

and suppose the prior is

$$p(a, b) = \exp(-a - b)I(a, b > 0).$$

You want to sample from the posterior $p(a, b|x)$

6.4 Markov Chain Monte Carlo Methods

Exponential Example: Gibbs Sampling Algorithm

Conditional distributions

$$\begin{aligned} p(\mathbf{x}|a,b) &= \prod_{i=1}^{n} p(x_i|a,b) \\ &= \prod_{i=1}^{n} ab\exp(-abx_i) \\ &= (ab)^n \exp\left(-ab\sum_{i=1}^{n} x_i\right). \end{aligned}$$

The function is symmetric for *a* and *b*, so we only need to derive *p(a|x, b)*.

This conditional distribution satisfies

$$\begin{aligned} p(a|\mathbf{x},b) &\propto_a p(a,b,\mathbf{x}) \\ &= p(\mathbf{x}|a,b)p(a,b) \\ &= \text{fill in full details for homework} \end{aligned}$$

R Code GS1

6.4 Markov Chain Monte Carlo Methods

Toy Example: Gibbs Sampling Algorithm

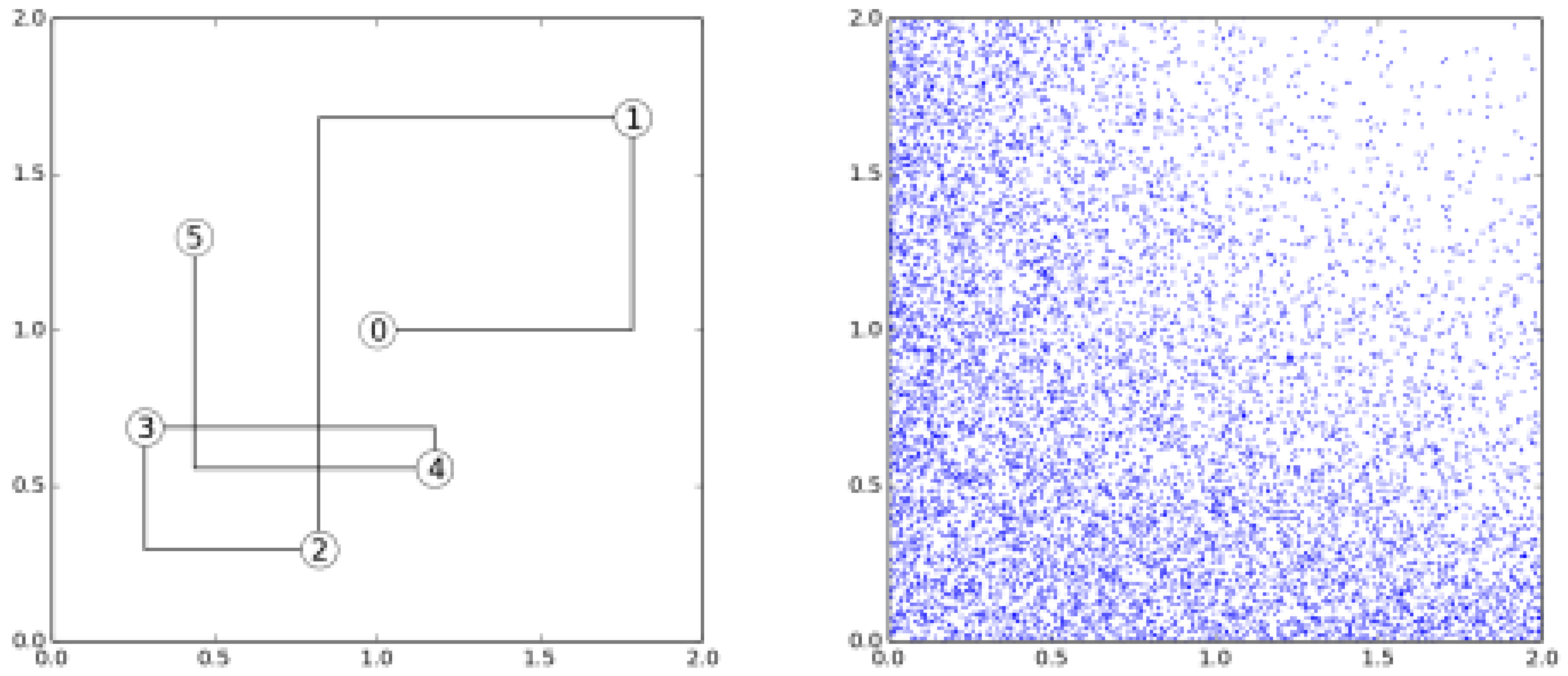


Figure 1: (Left) Schematic representation of the first 5 Gibbs sampling iterations/sweeps/scans. (Right) Scatterplot of samples from 10^4 Gibbs sampling iterations.

6.4 Markov Chain Monte Carlo Methods

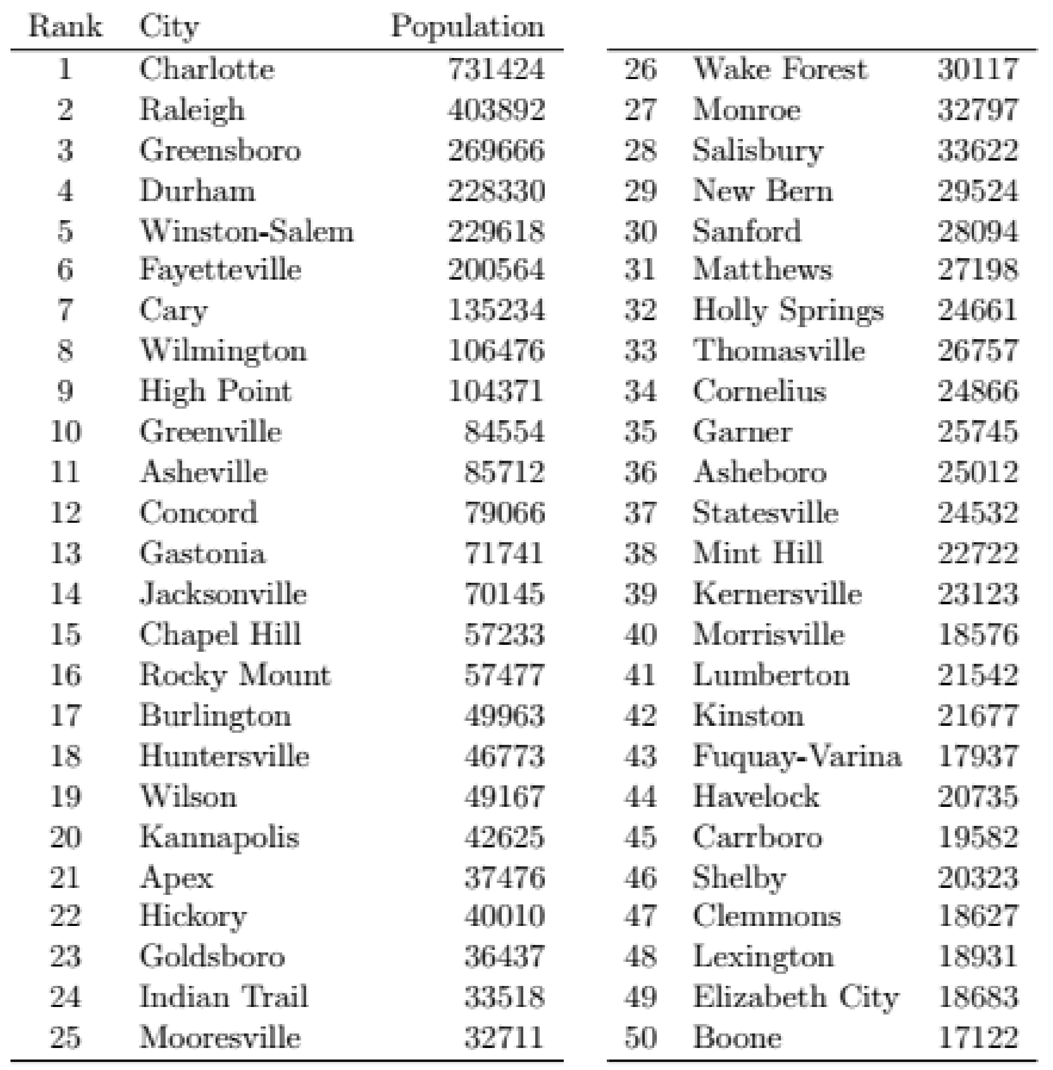
Power Law Example: Gibbs Sampling Algorithm

So, in order to use the Gibbs sampling algorithm to sample from the posterior $p(\alpha, c|x_{1:n})$, we initialize α and c, and then alternately update them by sampling:

$$\begin{aligned} \alpha|c, x_{1:n} &\sim \text{Gamma}\left(n+1, \sum \log x_i - n \log c\right) \\ c|\alpha, x_{1:n} &\sim \text{Mono}(n\alpha + 1, x_*). \end{aligned}$$

Initializing at $\alpha = 1$ and $c = 100$, we run the Gibbs sampler for $N = 10^3$ iterations on the 50 data points from Table 1, giving us a sequence of samples

$$(\alpha_1, c_1), \ldots, (\alpha_N, c_N).$$



Rank	City	Population	Rank	City	Population
1	Charlotte	731424	26	Wake Forest	30117
2	Raleigh	403892	27	Monroe	32797
3	Greensboro	269666	28	Salisbury	33622
4	Durham	228330	29	New Bern	29524
5	Winston-Salem	229618	30	Sanford	28094
6	Fayetteville	200564	31	Matthews	27198
7	Cary	135234	32	Holly Springs	24661
8	Wilmington	106476	33	Thomasville	26757
9	High Point	104371	34	Cornelius	24866
10	Greenville	84554	35	Garner	25745
11	Asheville	85712	36	Asheboro	25012
12	Concord	79066	37	Statesville	24532
13	Gastonia	71741	38	Mint Hill	22722
14	Jacksonville	70145	39	Kernersville	23123
15	Chapel Hill	57233	40	Morrisville	18576
16	Rocky Mount	57477	41	Lumberton	21542
17	Burlington	49963	42	Kinston	21677
18	Huntersville	46773	43	Fuquay-Varina	17937
19	Wilson	49167	44	Havelock	20735
20	Kannapolis	42625	45	Carrboro	19582
21	Apex	37476	46	Shelby	20323
22	Hickory	40010	47	Clemmons	18627
23	Goldsboro	36437	48	Lexington	18931
24	Indian Trail	33518	49	Elizabeth City	18683
25	Mooresville	32711	50	Boone	17122

Table 1: Populations of the 50 largest cities in the state of North Carolina, USA.

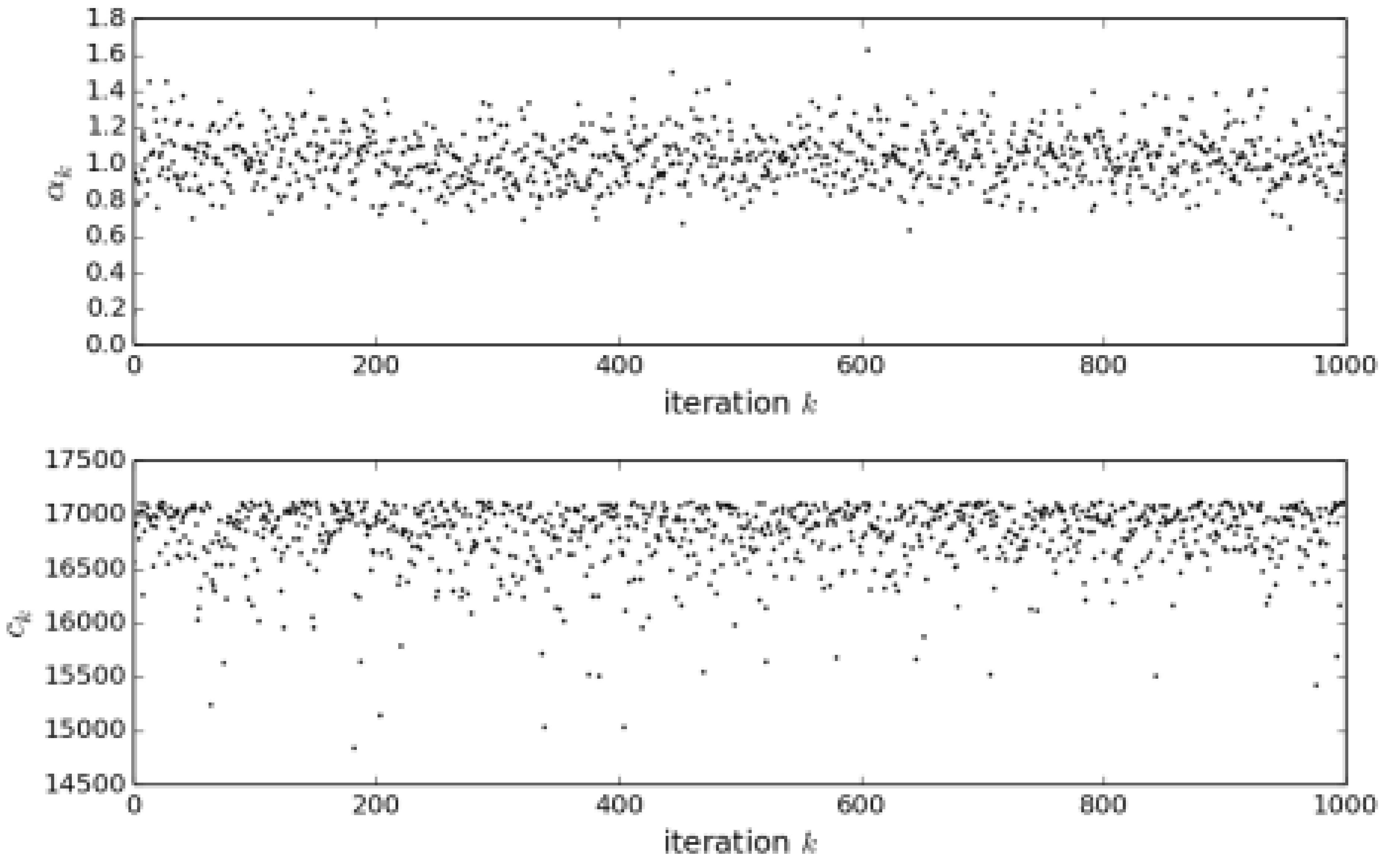
6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm: Results

Traceplots

A traceplot simply shows the sequence of samples, for instance $\alpha_1, \ldots, \alpha_N$, or $c_1, \ldots, c_N$.

*Traceplots are a simple but very useful way to visualize how the sampler is behaving. The traceplots in **Figure 2(a)** look very healthy—the sampler does not appear to be getting stuck anywhere.*



(a) Traceplots of α (top) and c (bottom).

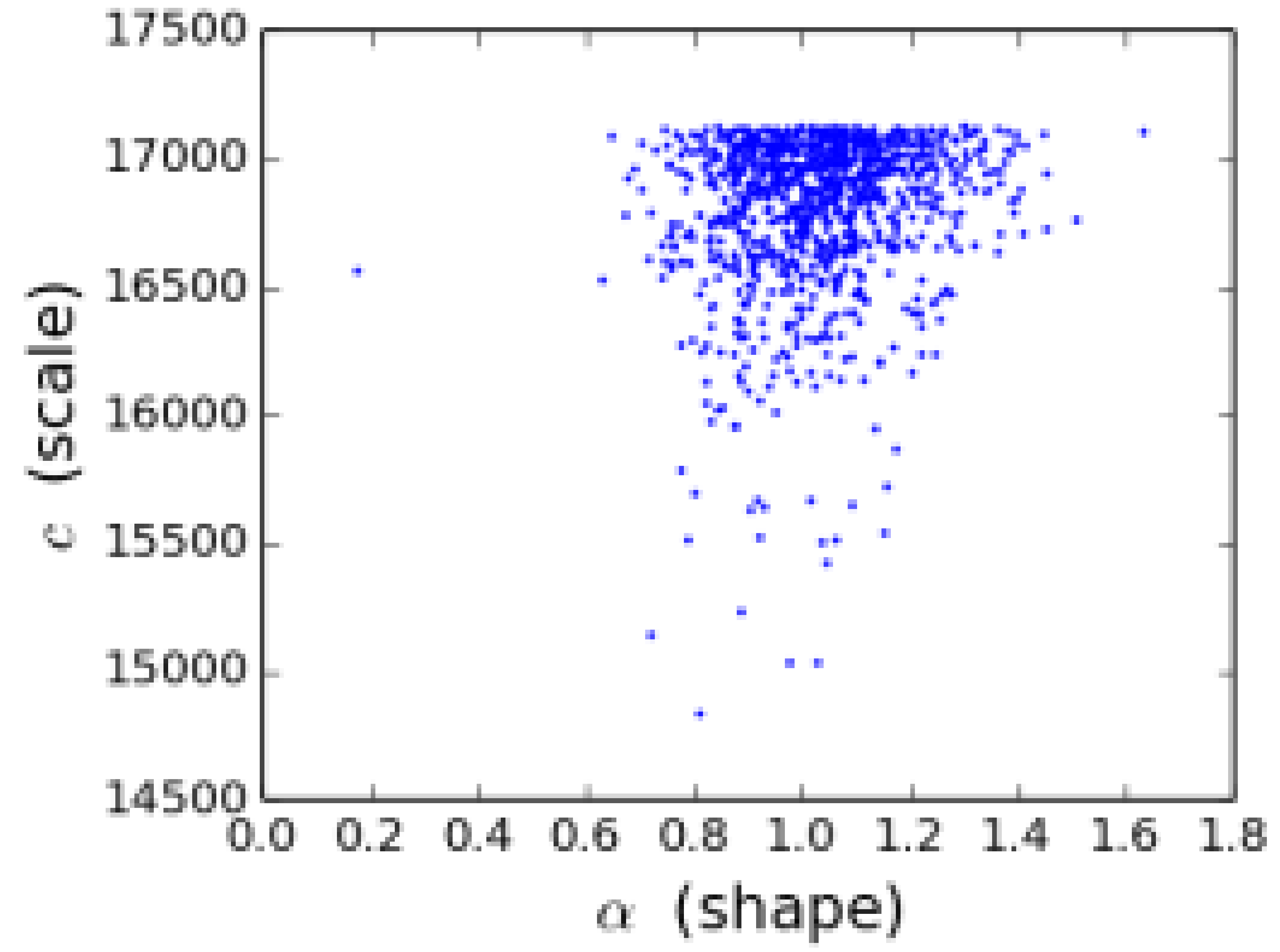
6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm: Results

Scatterplot

*The scatterplot in **panel (b)** shows us what the posterior distribution $p(\alpha; c|x_{1:n})$ looks like.*

The smallest city in our data set is Boone, with a population of 17,122, and the posterior on c is quite concentrated just under this value, which makes sense since c represents the cutoff point in the sampling process.



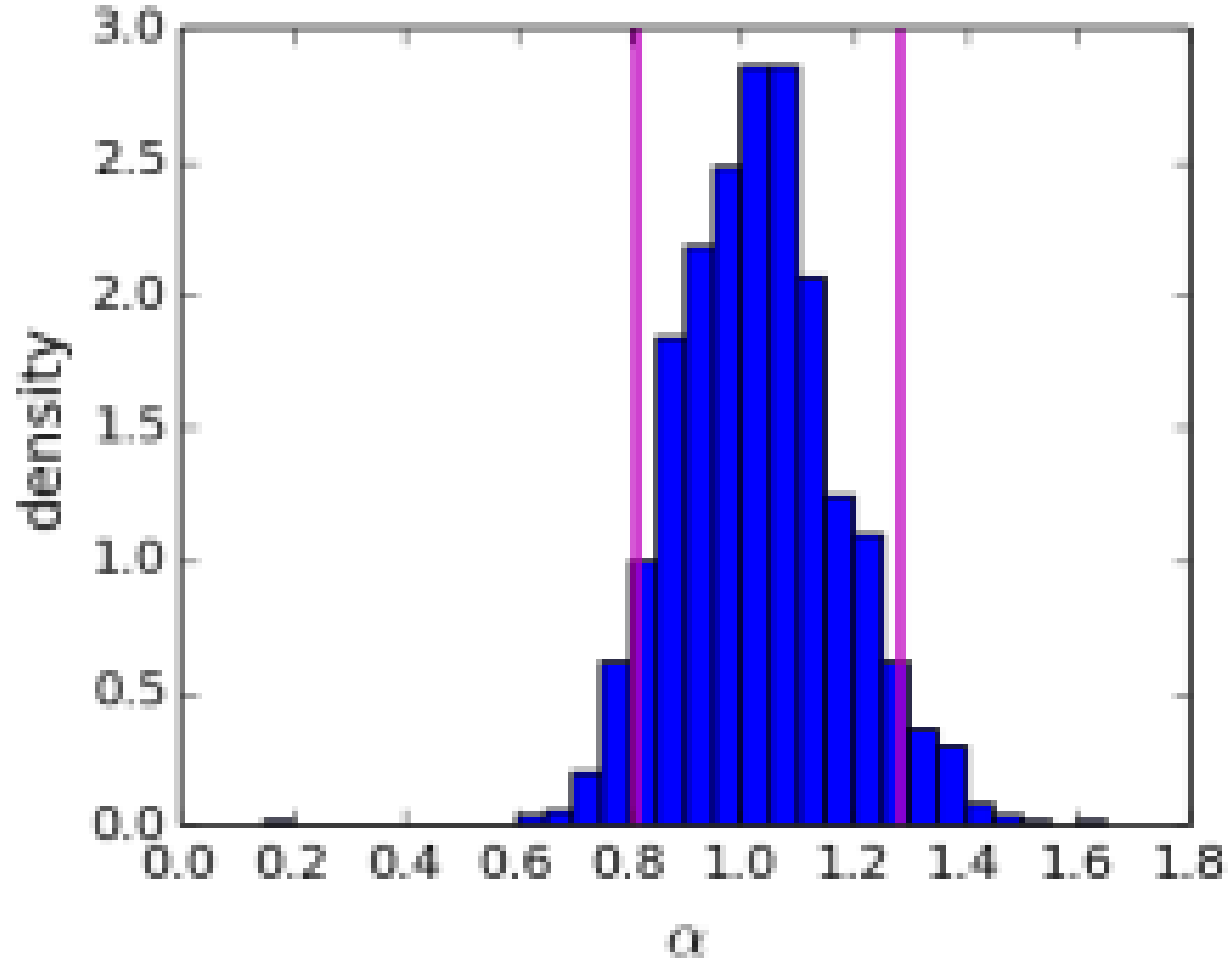
(b) Scatterplot of samples.

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm: Results

Estimated density (using Histogram and CI)

We are primarily interested in the posterior on α, since it tells us the scaling relationship between the size of cities and their probability of occurring. By making a histogram of the samples $\alpha_{1,\dots,}\alpha_N$, we can estimate the posterior density $p(\alpha|x_{1:n})$. The two vertical lines indicate the lower and upper boundaries of an (approximate) 90% credible interval that is, an interval that contains 90% of the posterior probability



(c) Estimated density of $\alpha|x_{1:n}$.

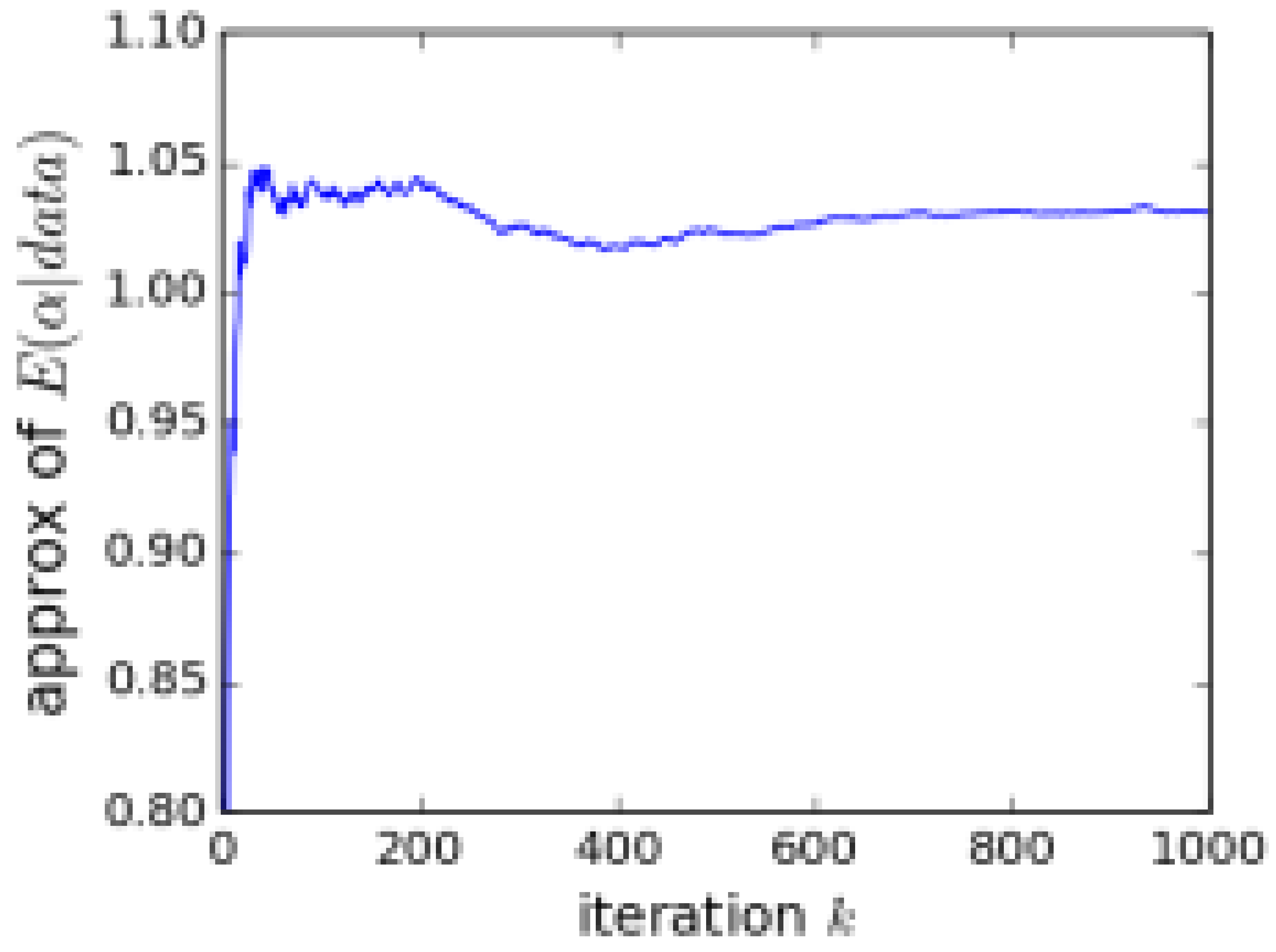
6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm: Results

Running averages

Panel (d) shows the running average. In addition to traceplots, running averages such as this are a useful heuristic for visually assessing the convergence of the Markov chain.

The running average shown in this example still seems to be meandering about a bit, suggesting that the sampler needs to be run longer (but this would depend on the level of accuracy desired).



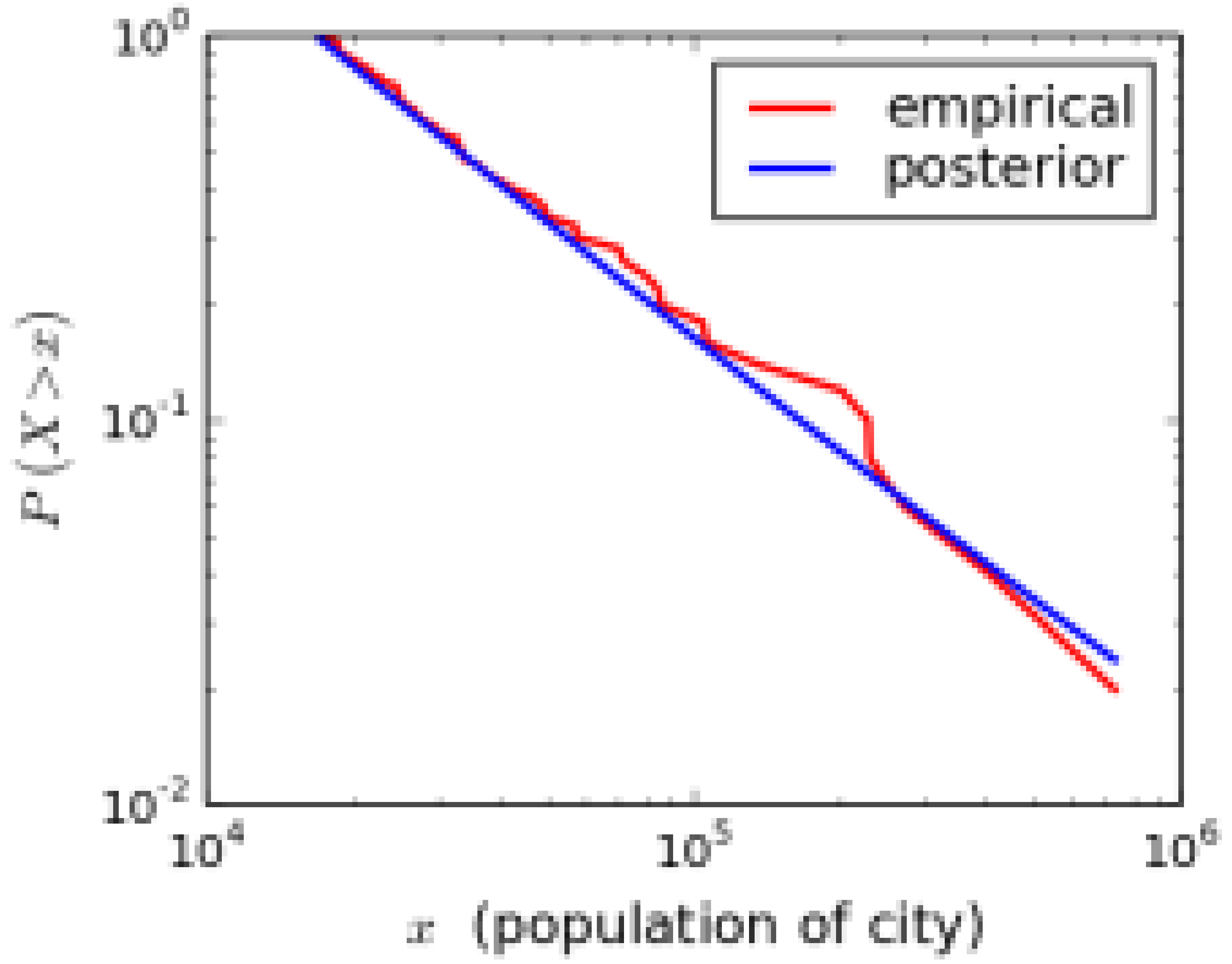
(d) $\frac{1}{k}\sum_{i=1}^{k}\alpha_i$ for $k = 1, \ldots, N$.

6.4 Markov Chain Monte Carlo Methods

Gibbs Sampling Algorithm: Results

Panel (e) is particular to this example. Power law distributions are often displayed by plotting their survival function S(x)- that is, one minus the c.d.f.,

$S(x) = P(X > x) = 1 - P(X \leq x)$ - on a log-log plot, since $S(x) = (c=x)^{\alpha}$ for the Pareto(α, c) distribution and on a log-log plot this appears as a line with slope $-\alpha$.



(e) Empirical vs posterior survival function.

6.4 Markov Chain Monte Carlo Methods

Metropolis-Hastings Algorithm

- The Metropolis-Hastings Algorithm is appropriate for those probabilistic models where we cannot directly sample the so-called next state probability distribution, such as the conditional probability distribution used by Gibbs Sampling.
- The Metropolis-Hastings algorithm involves using a surrogate or proposal probability distribution that is sampled (sometimes called the kernel), then an acceptance criterion that decides whether the new sample is accepted into the chain or discarded.

They are based on a Markov chain whose dependence on the predecessor is split into two parts: a proposal and an acceptance of the proposal. The proposals suggest an arbitrary next step in the trajectory of the chain and the acceptance makes sure the appropriate limiting direction is maintained by rejecting unwanted moves of the chain.

6.4 Markov Chain Monte Carlo Methods

Metropolis-Hastings Algorithm

- The acceptance criterion is probabilistic based on how likely the proposal distribution differs from the true next-state probability distribution.

- The Metropolis-Hastings Algorithm is a more general and flexible Markov Chain Monte Carlo algorithm, subsuming many other methods.

- For **example**, if the next-step conditional probability distribution is used as the proposal distribution, then the Metropolis-Hastings is generally equivalent to the Gibbs Sampling Algorithm.

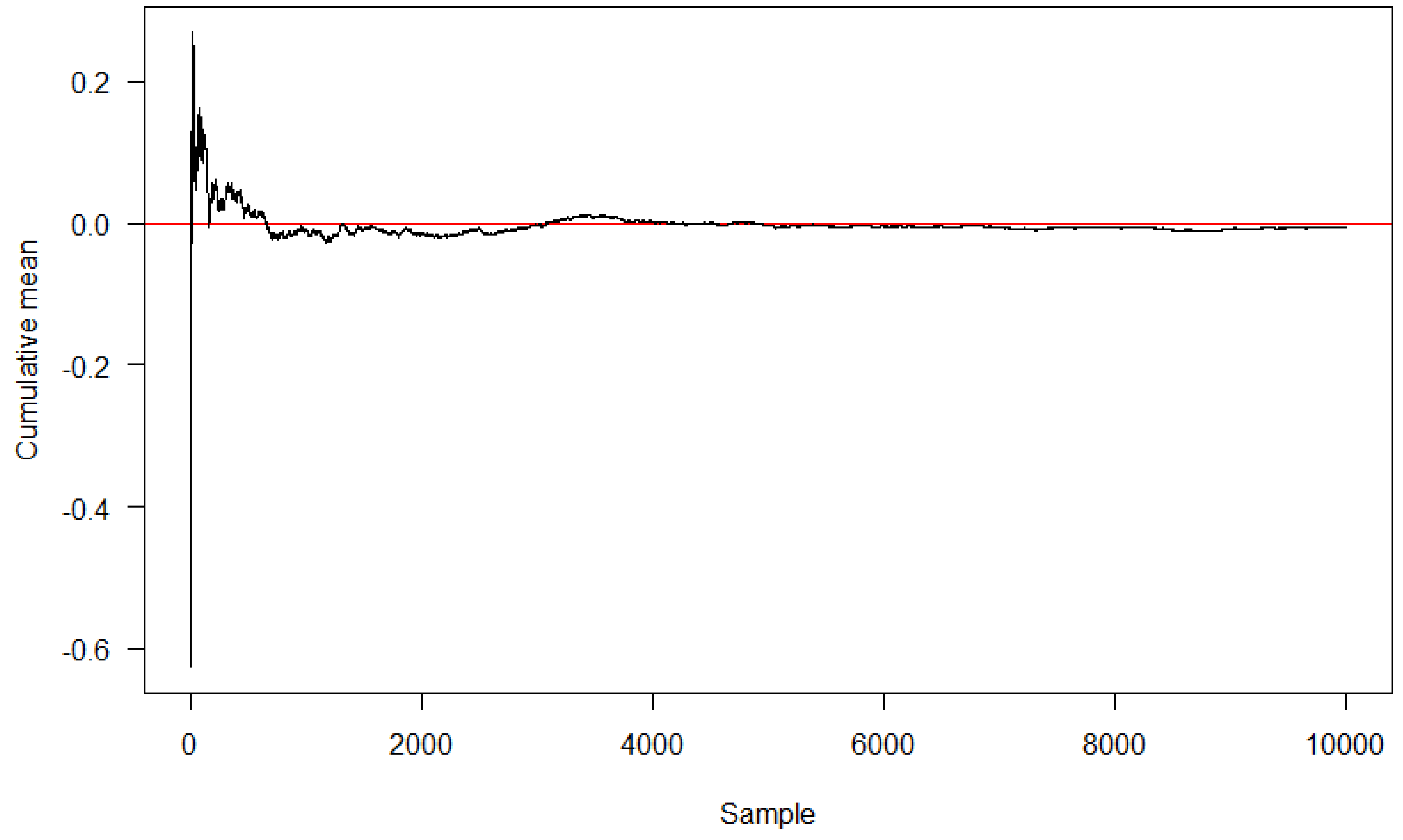
If a symmetric proposal distribution is used like a Gaussian, the algorithm is equivalent to another MCMC method called the Metropolis algorithm.

6.4 Markov Chain Monte Carlo Methods

Simple Example: Metropolis-Hastings Algorithm

Consider estimating the mean of a normal distribution with mean and standard deviations (use parameters corresponding to a standard normal)

```
#As an example, consider estimating the mean of a normal distribution
with mean m and standard deviation s (here use parameters corresponding
to a standard normal):

m <- 0
s <- 1

#We can easily sample from this distribution using the rnorm function:

set.seed(1)
samples <- rnorm(10000, m, s)

#The mean of the samples is very close to the true mean (zero):
mean(samples)

#replicates 1000
summary(replicate(1000, mean(rnorm(10000, m, s))))

#This function computes the cumulative mean
cummean <- function(x)
  cumsum(x) / seq_along(x)

#Here is the convergence towards the true mean (red line at 0).
plot(cummean(samples), type="l", xlab="Sample", ylab="Cumulative mean",
     panel.first=abline(h=0, col="red"), las=1)
```

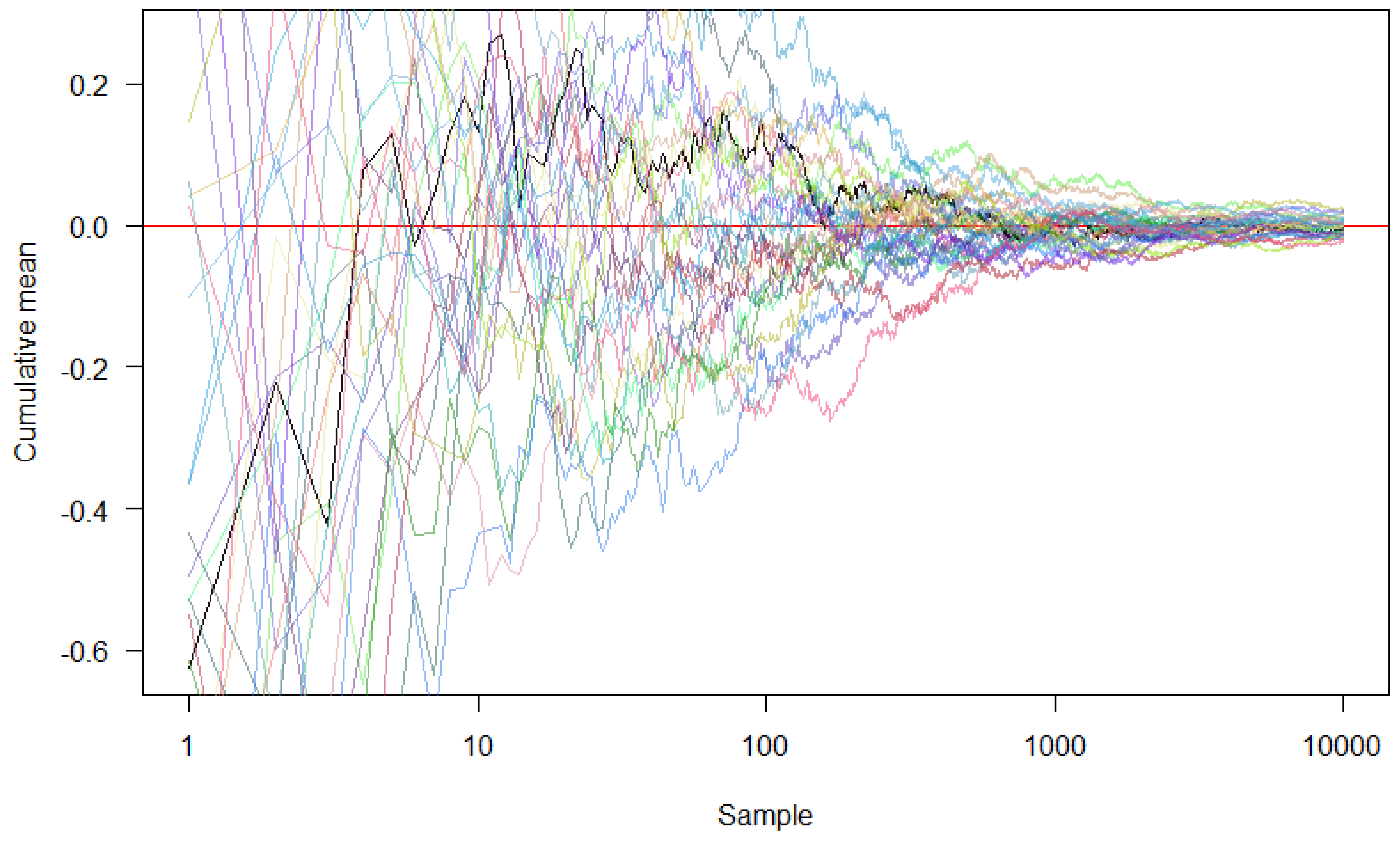


6.4 Markov Chain Monte Carlo Methods

Simple Example: Metropolis-Hastings Algorithm

Transforming the x axis onto a log scale and showing another 30 random approaches

```
set.seed(1)
plot(cummean(samples), type="l",
xlab="Sample", ylab="Cumulative
mean", panel.first=abline(h=0,
col="red"), las=1, log="x") for
(i in seq_len(30))
lines(cummean(rnorm(10000, m,
s)), col=rgb(runif(1), runif(1),
runif(1), .5))
```

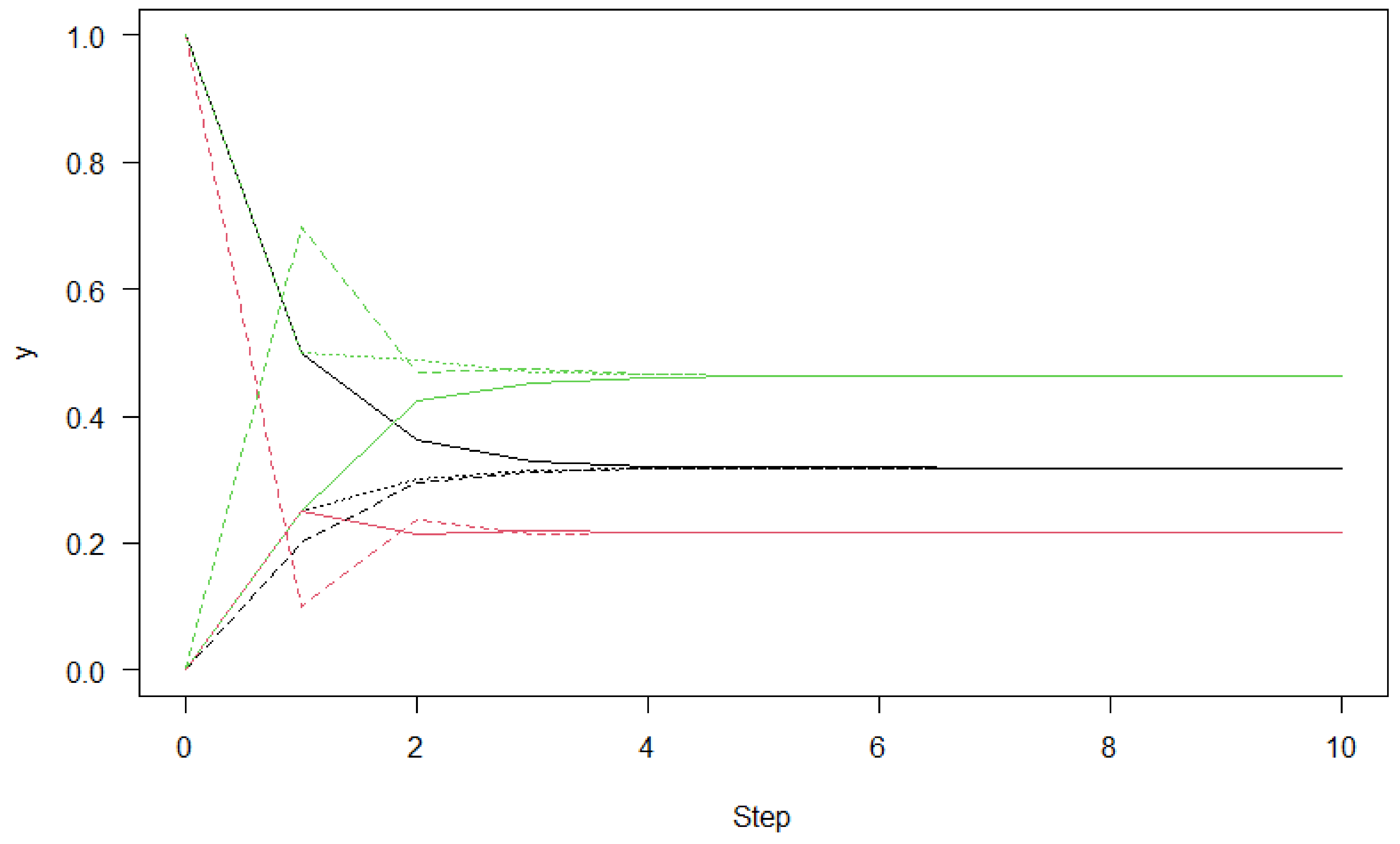


6.4 Markov Chain Monte Carlo Methods

MCMC Example: Metropolis-Hastings Algorithm

Suppose that we have a three-state Markov process. Let P be the transition probability matrix for the chain.

```
Refer R Code MCMC1
```



The graph shows that regardless of the starting distribution, there is a 32\% chance of the chain being in state 1 after about 10 or more iterations *regardless of where it started*. So, knowing about the state of this chain at one point in time gives you information about where it is likely to be for only a few steps.

6.4 Markov Chain Monte Carlo Methods

MCMC Example: Metropolis-Hastings Algorithm

MCMC sampling in 1d (single parameter) problems

```
Refer R Code MCMC2
```

The algorithm proceeds as follows.

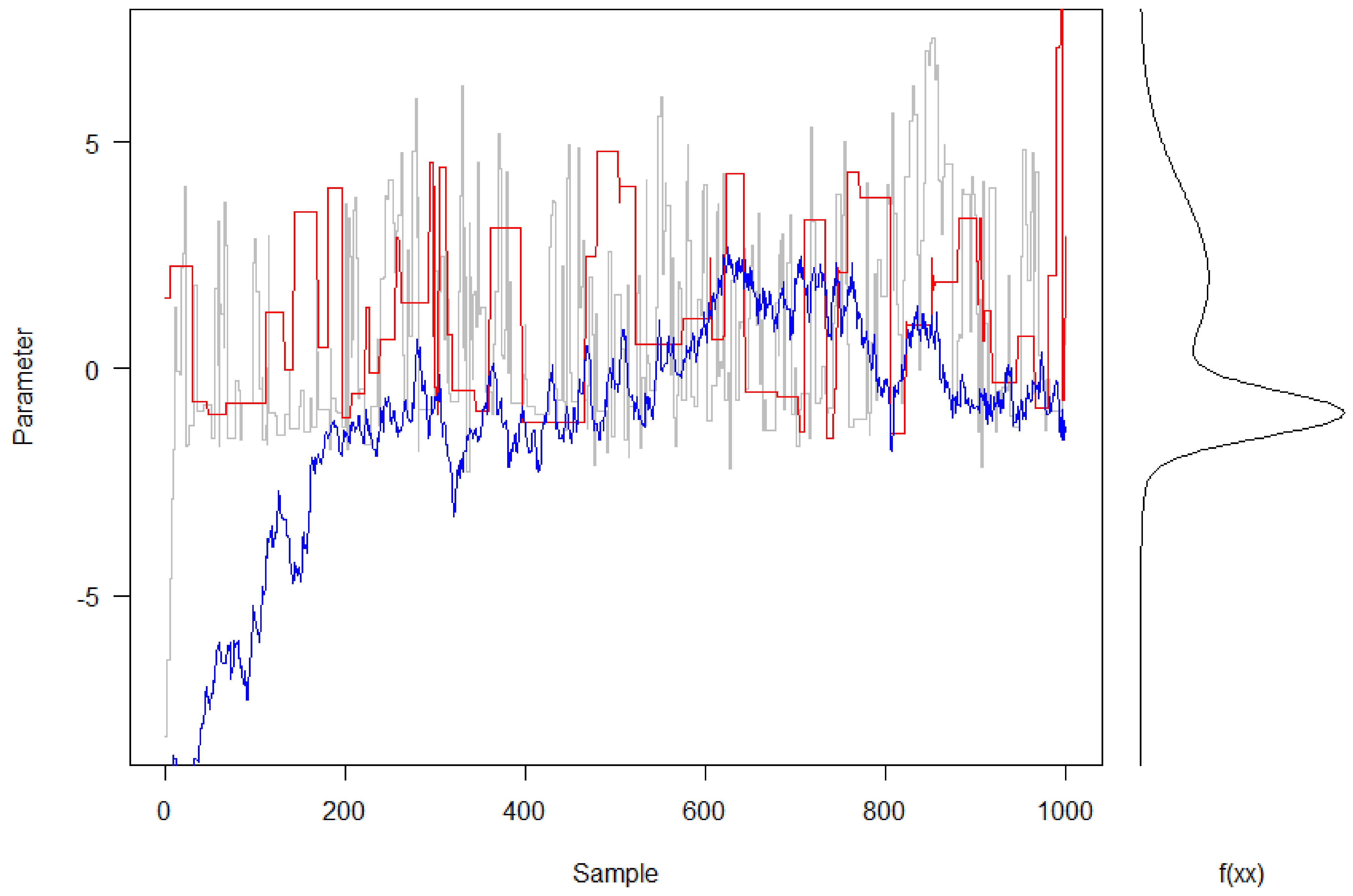
1. Start in some state x_t.
2. Propose a new state x’.
3. Compute the “acceptance probability”.
4. Draw some uniformly distributed random number u from $[0,1]$; if $u < \alpha$ accept the point, setting $x*{t+1}* = *x^\prime$. Otherwise reject it and set $x*{t+1} = x_t$.

6.4 Markov Chain Monte Carlo Methods

MCMC Example: Metropolis-Hastings Algorithm

MCMC sampling in 1d (single parameter) problems

Refer R Code MCMC2



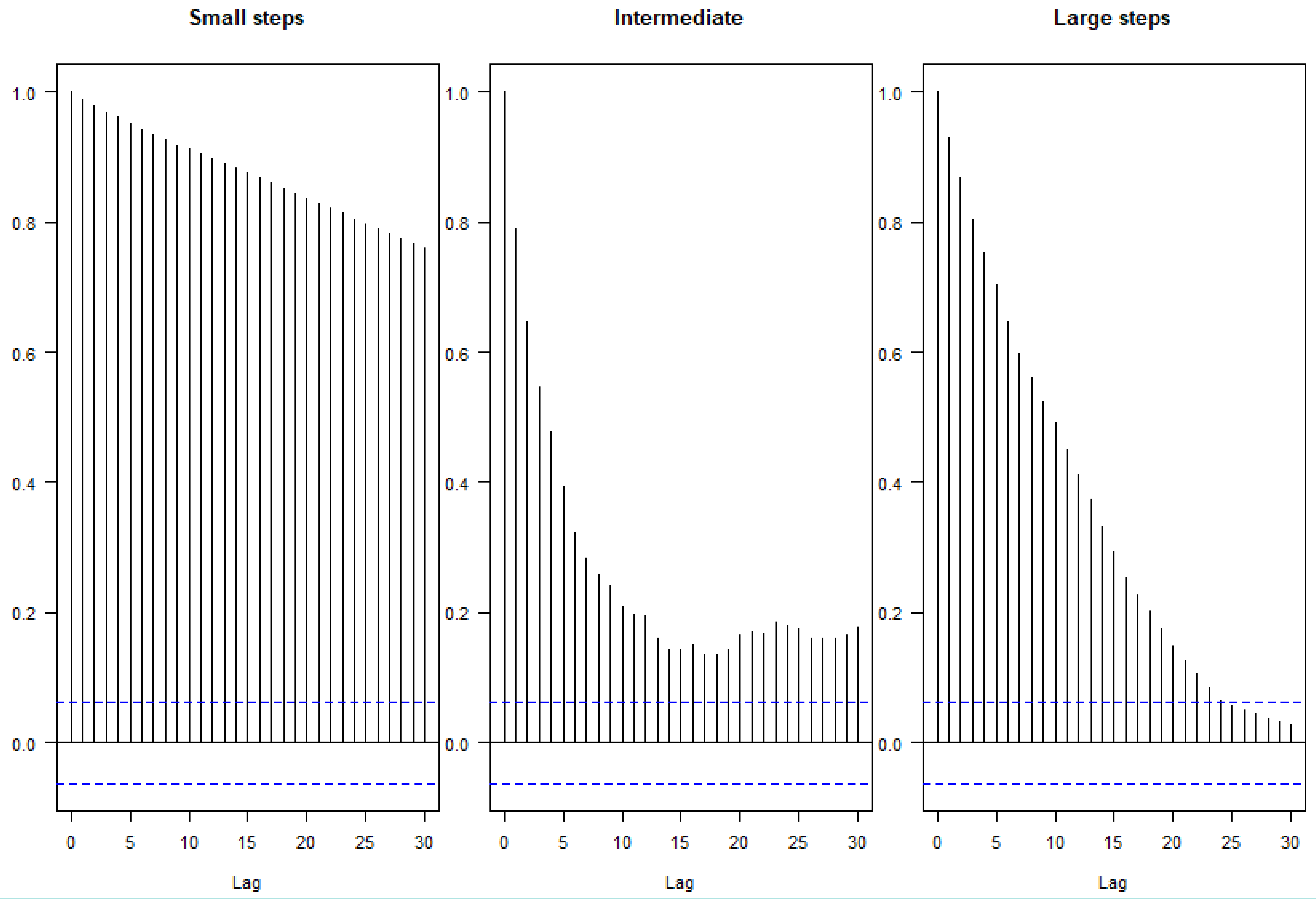
- The original (grey line) trace is bouncing around quite freely.
- In contrast, the red trace (large proposal moves) is suggesting terrible spaces in probability space and rejecting most of them. This means it tends to stay put for along time at once space.
- The blue trace proposes small moves that tend to be accepted, but it moves following a random walk for most of the trajectory. It takes hundreds of iterations to even reach the bulk of the probability density.

6.4 Markov Chain Monte Carlo Methods

MCMC Example: Metropolis-Hastings Algorithm

MCMC sampling in 1d (single parameter) problems

Refer R Code MCMC2



You can see the effect of different proposal steps in the autocorrelation among subsequent parameters – these plots show the decay in autocorrelation coefficient between steps of different lags, with the blue lines indicating statistical independence.

Summary

✓Compared to real-world data, synthetic data generation is faster, more flexible, and more scalable.

✓By adjusting parameters, it can also be an effective way to model and generate data that does not exist out in the real world.

✓Synthetic data allows data scientists to feed machine learning models with data to represent any situation.

✓Synthetic test data can reflect ‘what if’ scenarios, making it an ideal way to test a hypothesis or model multiple outcomes.

✓Synthetic data gives data scientists a way to do new, innovative things that are impossible with real-world data alone, feeding the models that will affect the way we all live in our data-driven future.

Summary

- ✓Monte Carlo sampling is not effective and may be intractable for high-dimensional probabilistic models.
- ✓Markov Chain Monte Carlo provides an alternate approach to random sampling a high-dimensional probability distribution where the next sample is dependent upon the current sample.
- ✓Gibbs Sampling and the more general Metropolis-Hastings algorithm are the two most common approaches to Markov Chain Monte Carlo sampling.

اونيۏرسيتي مليسيا ڤهڠ
UNIVERSITI MALAYSIA PAHANG