Stochastic Methods

INF SP 2024

Francisco Richter

Francisco Richter
richtf@usi.ch

Contents

Preliminaries

1.1 Randomness

Definition 1.1. Randomness refers to the inherent unpredictability and lack of pattern in events. Key characteristics include:

- **Unpredictability**: The inability to forecast future outcomes based on past events.
- **Patternlessness**: The absence of any discernible regularity or order.
- **Statistical Regularity**: The tendency to conform to predictable statistical properties over a large number of trials.
- **Independence**: The occurrence of one event does not influence the occurrence of another.
- **Reproducibility in Aggregates**: Consistent statistical properties when events are considered in large groups or aggregates.

1.2 Random Variable

Consider an experiment with a sample space S on which probabilities are defined. A random variable X is a function that assigns a real value to each outcome of the experiment. For any set of real numbers C, the probability that X will have a value that is contained in the set C is equal to the probability that the outcome of the experiment is contained in $X^{-1}(C)$. That is,

$$P\{X \in C\} = P\{X^{-1}(C)\},$$

where $X^{-1}(C)$ is the event consisting of all outcomes $s \in S$ such that $X(s) \in C$.
The distribution function F of the random variable X is defined for all real numbers by

$$F(x) = P\{X \leq x\} = P\{X \in (-\infty, x]\}.$$

A probability distribution describes how probabilities are distributed over the values of a random variable, providing the probabilities of occurrence of different possible outcomes.

Example. Geometric Distribution
Describes the number of trials needed to get the first success in a sequence of independent Bernoulli trials. The PMF is given by:

$$P(X = k) = (1 - p)^{k-1}p,$$

where k is the number of trials until the first success, and p is the probability of success on each trial. *Example: The probability of getting the first head in a series of coin flips.*

Example. Binomial Distribution
Gives the probability of observing a specific number of successes in a fixed number of independent Bernoulli trials. The PMF is:

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k},$$

where n is the number of trials, k is the number of successes, and p is the probability of success on each trial.
Example: The probability of getting exactly three heads in five tosses of a fair coin.

Example. Poisson Distribution
Describes the probability of a given number of events happening in a fixed interval of time or space. The PMF is:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!},$$

where k is the number of events, λ is the average number of events per interval.
Example: The number of cars passing through a checkpoint in an hour.

Example. Exponential Distribution
Describes the time between events in a Poisson point process. The PDF is:

$$f(x) = \lambda e^{-\lambda x},$$

for $x \geq 0$, where λ is the rate parameter.
Example: The amount of time until the next earthquake occurs in a given region.

Example. Normal Distribution
Describes how the values of a variable are distributed. The PDF is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

where μ is the mean and σ is the standard deviation.
Example: The distribution of heights of adult men in a specific population.

1.3 Expectation

The expectation (or expected value) of a random variable is a measure of the central tendency of its probability distribution. It is denoted by $E[X]$ for a random variable X and is calculated as:

$$E[X] = \sum_x xP(X = x)$$

for discrete random variables, and

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

for continuous random variables, where $P(X = x)$ is the probability mass function for discrete variables and $f(x)$ is the probability density function for continuous variables.

1.4 Variance

Variance measures the dispersion of a random variable's values around its mean. It is denoted by Var(X) or σ_X^2 and is calculated as:

$$\text{Var}(X) = E[(X - \mu)^2] = E[X^2] - (E[X])^2,$$

where $\mu = E[X]$ is the mean (or expectation) of X.
Let's calculate the expectation and variance of a binomial distribution with parameters n and p, where n is the number of trials and p is the probability of success.
Expectation:

$$E[X] = \sum_{k=0}^{n} k\binom{n}{k}p^k(1-p)^{n-k}.$$

We use the binomial theorem and properties of binomial coefficients to simplify this expression, recognizing that this is equivalent to np by considering the derivative of the binomial expansion $(p + (1 - p))^n$.
Variance:

$$\text{Var}(X) = E[X^2] - (E[X])^2.$$

First, calculate $E[X^2]$:

$$E[X^2] = \sum_{k=0}^{n} k^2\binom{n}{k}p^k(1-p)^{n-k}.$$

This requires using the binomial theorem and properties of binomial coefficients, similar to the expectation but involves a more complex manipulation. Eventually, we find that $E[X^2] = np(1 - p) + n^2p^2$. Substituting $E[X] = np$ into the variance formula gives:

$$\text{Var}(X) = np(1-p) + n^2p^2 - (np)^2 = np(1-p).$$

1.5 Conditional Probability

Recall that for any two events E and F, the conditional probability of E given F is defined, as long as $P(F) > 0$, by

$$P(E|F) = \frac{P(EF)}{P(F)}$$

Hence, if X and Y are discrete random variables, then it is natural to define the conditional probability mass function of X given that $Y = y$, by

$$p_{X|Y}(x|y) = P\{X = x|Y = y\} = \frac{P\{X = x, Y = y\}}{P\{Y = y\}} = \frac{p(x,y)}{p_Y(y)}$$

where $p(x,y)$ is the joint probability mass function of X and Y, and $p_Y(y)$ is the marginal probability mass function of Y.
Let E denote an arbitrary event and define the indicator random variable X by

$$X = \begin{cases} 1, & \text{if } E \text{ occurs,} \\ 0, & \text{if } E \text{ does not occur.} \end{cases}$$

It follows from the definition of X that

$$E[X] = P(E),$$

$$E[X|Y = y] = P(E|Y = y),$$

for any random variable Y.
Therefore,

$$P(E) = \sum_y P(E|Y = y)P(Y = y), \text{ if } Y \text{ is discrete}$$

$$P(E) = \int_{-\infty}^{\infty} P(E|Y = y)f_Y(y)\,dy, \text{ if } Y \text{ is continuous}$$

where $f_Y(y)$ is the probability density function of Y.

Example. If X_1 and X_2 are independent binomial random variables with respective parameters (n_1, p) and (n_2, p), calculate the conditional probability mass function of X_1 given that $X_1 + X_2 = m$.

Solution: With $q = 1 - p$,

$$\begin{aligned} P\{X_1 = k|X_1 + X_2 = m\} &= \frac{P\{X_1 = k, X_1 + X_2 = m\}}{P\{X_1 + X_2 = m\}} \\ &= \frac{P\{X_1 = k, X_2 = m - k\}}{P\{X_1 + X_2 = m\}} \\ &= \frac{P\{X_1 = k\}P\{X_2 = m - k\}}{P\{X_1 + X_2 = m\}} \\ &= \frac{\binom{n_1}{k} p^k q^{n_1-k} \binom{n_2}{m-k} p^{m-k} q^{n_2-(m-k)}}{\binom{n_1+n_2}{m} p^m q^{n_1+n_2-m}} \end{aligned}$$

where we have used that $X_1 + X_2$ is a binomial random variable with parameters $(n_1 + n_2, p)$ (see Example 2.44). Thus, the conditional probability mass function of X_1, given that $X_1 + X_2 = m$, is

$$P\{X_1 = k | X_1 + X_2 = m\} = \frac{\binom{n_1}{k}\binom{n_2}{m-k}}{\binom{n_1+n_2}{m}}$$

The distribution is known as the hypergeometric distribution. It is the distribution of the number of blue balls that are chosen when a sample of m balls is randomly chosen from an urn that contains n_1 blue and n_2 red balls.

Example. If X and Y are independent Poisson random variables with respective means λ_1 and λ_2, calculate the conditional expected value of X given that $X + Y = n$.

Solution: Let us first calculate the conditional probability mass function of X given that $X + Y = n$. We obtain

$$\begin{aligned}
P\{X = k | X + Y = n\} &= \frac{P\{X = k, X + Y = n\}}{P\{X + Y = n\}} \\
&= \frac{P\{X = k, Y = n - k\}}{P\{X + Y = n\}} \\
&= \frac{P\{X = k\}P\{Y = n - k\}}{P\{X + Y = n\}} \\
&= \frac{e^{-\lambda_1}\frac{\lambda_1^k}{k!}e^{-\lambda_2}\frac{\lambda_2^{n-k}}{(n-k)!}}{e^{-(\lambda_1+\lambda_2)}\frac{(\lambda_1+\lambda_2)^n}{n!}} \\
&= \frac{n!}{(n-k)!k!}\frac{\lambda_1^k\lambda_2^{n-k}}{(\lambda_1+\lambda_2)^n} \\
&= \binom{n}{k}\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)^k\left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)^{n-k}
\end{aligned}$$

In other words, the conditional distribution of X given that $X + Y = n$, is the binomial distribution with parameters n and $\frac{\lambda_1}{\lambda_1+\lambda_2}$. Hence,

$$E\{X | X + Y = n\} = n\frac{\lambda_1}{\lambda_1 + \lambda_2}$$

This result demonstrates the conditional expectation of X given the sum $X + Y$, showing that it follows a binomial distribution in the context of the given Poisson random variables.

Example. Suppose the joint density of X and Y is given by

$$f(x, y) = \begin{cases} 6xy(2 - x - y), & 0 < x < 1, 0 < y < 1, \\ 0, & \text{otherwise.} \end{cases}$$

Compute the conditional expectation of X given that $Y = y$, where $0 < y < 1$.

Solution: We first compute the conditional density

$$f_{X|Y}(x|y) = \frac{f(x, y)}{f_Y(y)}$$

$$= \frac{6xy(2-x-y)}{\int_0^1 6xy(2-x-y)\,dx}$$
$$= \frac{6xy(2-x-y)}{y(4-3y)}$$
$$= \frac{6x(2-x-y)}{4-3y}$$

Hence,

$$E[X|Y=y] = \int_0^1 x \cdot \frac{6x(2-x-y)}{4-3y}\,dx$$
$$= \frac{(2-y)^2 - \frac{6}{4}}{4-3y}$$
$$= \frac{5-4y}{8-6y}$$

This result provides the conditional expectation $E[X|Y=y]$, which is a function of y.

1.6 Moments

Definition: Moments
Moments are quantitative measures used to describe the shape of a probability distribution. The n-th moment of a random variable X about the mean is defined as:

$$\mu_n = E[(X-\mu)^n],$$

where $\mu = E[X]$ is the mean of X. The first moment about the mean is the mean itself, the second moment about the mean is the variance, and higher moments describe skewness, kurtosis, and other aspects of the distribution's shape.

Moment-Generating Function

Definition: Moment-Generating Function
The moment-generating function (MGF) of a random variable X is defined as:

$$M_X(t) = E[e^{tX}],$$

where t is a real number, and the expectation is taken over the probability distribution of X. The MGF, if it exists, uniquely determines the probability distribution of X and can be used to find all the moments of the distribution since the n-th moment of X is given by the n-th derivative of $M_X(t)$ evaluated at $t=0$:

$$\mu_n = \left.\frac{d^n M_X(t)}{dt^n}\right|_{t=0}.$$

Example:
Consider a random variable X that follows an exponential distribution with rate parameter λ. The MGF of X is:

$$M_X(t) = \int_0^\infty e^{tx}\lambda e^{-\lambda x}dx = \frac{\lambda}{\lambda-t},$$

for $t < \lambda$. The first derivative of $M_X(t)$ with respect to t, evaluated at $t = 0$, gives the mean (the first moment) of X:

$$E[X] = \left.\frac{dM_X(t)}{dt}\right|_{t=0} = \frac{1}{\lambda}.$$

Example. Consider a discrete uniform random variable X that can take on the values $1, 2, 3, \ldots, n$ with equal probability.
Task: Find the moment-generating function (MGF) of X, $M_X(t)$, and use it to compute the first and second moments ($E[X]$ and $E[X^2]$).
The MGF of a random variable X is defined as $M_X(t) = E[e^{tX}]$. For a discrete uniform distribution over $1, 2, \ldots, n$, this becomes:

$$M_X(t) = \frac{1}{n}\sum_{k=1}^{n} e^{tk} = \frac{1}{n}\frac{e^t(1 - e^{tn})}{1 - e^t}, \quad \text{for } t \neq 0.$$

To find the first and second moments, we differentiate $M_X(t)$ with respect to t and evaluate at $t = 0$:

- First moment ($E[X]$): $M_X'(0) = \frac{1}{2}(n + 1)$.
- Second moment ($E[X^2]$): $M_X''(0) = \frac{1}{6}n(n + 1)(2n + 1)$.

Markov Chains

2.1 Stochastic Processes

Definition 2.1. A **stochastic process** $\{X(t) : t \in T\}$ is a collection of random variables, where for each t in the index set T, $X(t)$ is a random variable. The index t often represents time, making $X(t)$ denote the state of the process at time t. The set T is known as the index set of the process. When T is countable, the process is said to be in discrete time. If T is an interval of the real line, the process is in continuous time.

This broad definition encompasses the wide variety of processes that evolve over time, where the evolution is driven by some probabilistic rules. Examples include the number of customers in a store at any given time, the amount of rainfall accumulated in a day, or the stock price of a company.

Definition 2.2. A Markov Chain is a stochastic process that satisfies the Markov property, meaning the future state depends only on the current state and not on the sequence of events that preceded it. Formally, for any set of states i, j and any time n, the transition probabilities satisfy:

$$P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \dots, X_0 = i_0\} = P\{X_{n+1} = j | X_n = i\} = P_{ij}.$$

Examples of Markov Chains include:

- **Random Walk Model:** An individual moves along a line, stepping right with probability p and left with probability $1 - p$. The state space is the set of integers $\mathbb{Z}$.
- **Gambling Model:** A gambler wins 1 with probability p or loses 1 with probability $1-p$, stopping when they go broke or reach N. The state space is $\{0, 1, \dots, N\}$, with 0 and N as absorbing states.
- **Weather Forecasting Model:** The weather is either rain."r "no rain"with transition probability α for rain to rain and β for no rain to rain.
- **Communications System:** A system transmits digits 0 or 1, with each digit being transmitted correctly with probability p. This can be modeled as a two-state Markov chain.
- **Mood Model:** An individual's mood is modeled as cheerful, so-so, or glum, with transitions dependent only on the current state, forming a three-state Markov chain.

The transition matrix P encapsulates the probabilities of moving from one state to another in one time step. For a Markov chain with m states, P is an $m \times m$ matrix where the element P_{ij} represents the probability of transitioning from state i to state j.

Example. Weather Forecasting Model
Suppose the chance of rain tomorrow depends only on whether or not it is raining today. If it rains today (state 0), it will rain tomorrow with probability α; if it does not rain today (state 1), it will rain tomorrow with probability β. The transition matrix P is given by:

$$P = \begin{pmatrix} \alpha & 1-\alpha \\ \beta & 1-\beta \end{pmatrix}$$

Example. Communications System
Consider a system transmitting digits 0 and 1. At each stage, there is a probability p that the digit will be transmitted correctly. Letting X_n denote the digit at stage n, we have a two-state Markov chain with transition matrix:

$$P = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$$

Example. Mood Model
On any given day, Gary's mood can be cheerful (C), so-so (S), or glum (G) with transition probabilities dependent on today's mood. The transition matrix P is:

$$P = \begin{pmatrix} 0{,}5 & 0{,}4 & 0{,}1 \\ 0{,}3 & 0{,}4 & 0{,}3 \\ 0{,}2 & 0{,}3 & 0{,}5 \end{pmatrix}$$

Example. Random Walk Model
A model for an individual walking on a line who at each point either steps right with probability p or left with probability $1-p$. For state i, the transition probabilities to $i+1$ and $i-1$ are:

$$P_{i,i+1} = p = 1 - P_{i,i-1}$$

For a finite state space version of the **Random Walk Model**, with states $\{0, 1, 2, \ldots, N\}$ where N is a boundary, the transition probabilities for moving from state i to $i+1$ (right) with probability p and to $i-1$ (left) with probability $1-p$ can be represented as:

$$P = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1-p & 0 & p & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1-p & 0 & p \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

Note: This matrix is modified to include absorbing states at 0 and N for illustration purposes.

Example. Gambling Model
A gambler either wins \$1 with probability p or loses \$1 with probability $1-p$. With absorbing states at 0 and N (broke or target fortune), the transition probabilities for states i to $i+1$ and $i-1$ are similar to the random walk:

$$P_{i,i+1} = p = 1 - P_{i,i-1}$$

Absorbing states:

$$P_{00} = P_{NN} = 1$$

In the **Gambling Model**, with the gambler starting with a stake of i dollars and with absorbing states at 0 and N, the transition probabilities can be written as:

$$P = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ q & 0 & p & \cdots & 0 & 0 \\ 0 & q & 0 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & q & 0 & p \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Where p is the probability of winning \$1 and $q = 1-p$ is the probability of losing \$1. Here, states 0 and N are absorbing, meaning if the gambler reaches these states, they stay there indefinitely.

Let's consider a Markov chain with three states. The transition matrix for this Markov chain in generic form is given by:

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}$$

where p_{ij} represents the probability of transitioning from state i to state j.
To explore the long-term behavior of the Markov chain, we investigate the eigenvalues and eigenvectors of the transition matrix P. The eigenvalues are solutions to the characteristic equation given by $\det(P - \lambda I) = 0$, and for each eigenvalue λ_i, the corresponding eigenvector v_i is found by solving the equation $(P - \lambda_i I)v_i = 0$.
If we are able to find three linearly independent eigenvectors, we can form a matrix V composed of these eigenvectors and a diagonal matrix D containing the eigenvalues such that:

$$V = \begin{pmatrix} | & | & | \\ v_1 & v_2 & v_3 \\ | & | & | \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}.$$

This allows us to express P as $P = VDV^{-1}$.
To find P^n, where n is a positive integer, we use the property of matrix powers in diagonalized form:

$$P^n = VD^nV^{-1},$$

where D^n is the diagonal matrix with the eigenvalues raised to the power of n:

$$D^n = \begin{pmatrix} \lambda_1^n & 0 & 0 \\ 0 & \lambda_2^n & 0 \\ 0 & 0 & \lambda_3^n \end{pmatrix}.$$

Diagonalization of the transition matrix facilitates efficient computation of its powers. This method provides a straightforward way to determine the state probabilities after any number of transitions, illuminating the Markov chain's long-term dynamics.

2.2 Chapman-Kolmogorov Equations

The n-*step transition probabilities*, denoted by $P_{ij}^{(n)}$, define the probability that a process in state i will transition to state j after n steps. Formally, for $n \geq 0$ and states i, j, we have:

$$P_{ij}^{(n)} = \Pr(X_{n+k} = j \mid X_k = i)$$

where $P_{ij}^{(1)} = P_{ij}$ corresponds to the one-step transition probability from state i to state j.
The Chapman-Kolmogorov equations describe how to compute the probabilities of transitioning from one state to another in a Markov chain over multiple steps. Given a Markov chain with states, the n-step transition probability from state i to state j, denoted as $P_{ij}^{(n)}$, is the probability of transitioning from i to j in n steps.
These probabilities can be computed using the Chapman-Kolmogorov equations, which are as follows:

$$P_{ij}^{(n+m)} = \sum_{k=0}^{\infty} P_{ik}^{(n)} P_{kj}^{(m)} \tag{2.1}$$

This equation states that the probability of moving from state i to state j in $n + m$ steps is the sum, over all possible intermediate states k, of the probability of moving from i to k in n steps and then from k to j in m steps.
In matrix notation, if $P^{(n)}$ denotes the matrix of n-step transition probabilities, then the Chapman-Kolmogorov equation can be expressed as:

$$P^{(n+m)} = P^{(n)} \cdot P^{(m)} \tag{2.2}$$

where the dot represents matrix multiplication. This formula is particularly useful for computing transition probabilities over multiple steps in a compact and efficient manner.

Example. Consider the weather as a two-state Markov chain, where state 0 represents rain and state 1 represents no rain. Given the transition probabilities $\alpha = 0{,}7$ for rain to rain and $\beta = 0{,}4$ for no rain to rain, we calculate the probability that it will rain four days from today, given that it is raining today.
The one-step transition probability matrix is:

$$P = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}4 & 0{,}6 \end{pmatrix}$$

To find the two-day transition probabilities, we compute $P^{(2)} = P^2$:

$$P^{(2)} = P \cdot P = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}4 & 0{,}6 \end{pmatrix} \cdot \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}4 & 0{,}6 \end{pmatrix} = \begin{pmatrix} 0{,}61 & 0{,}39 \\ 0{,}52 & 0{,}48 \end{pmatrix}$$

For the four-day transition probabilities, we compute $P^{(4)} = (P^{(2)})^2$:

$$P^{(4)} = (P^{(2)})^2 = \begin{pmatrix} 0{,}61 & 0{,}39 \\ 0{,}52 & 0{,}48 \end{pmatrix} \cdot \begin{pmatrix} 0{,}61 & 0{,}39 \\ 0{,}52 & 0{,}48 \end{pmatrix} = \begin{pmatrix} 0{,}5749 & 0{,}4251 \\ 0{,}5668 & 0{,}4332 \end{pmatrix}$$

Therefore, the probability that it will rain four days from today, given that it is raining today (i.e., the transition from state 0 to state 0 in four steps), is $P_{00}^{(4)} = 0{,}5749$

Example. Weather Dependence as a Four-State Markov Chain
Consider the weather as a four-state Markov chain with the following states based on the weather conditions of today and yesterday:

- State 0: It rained both today and yesterday.
- State 1: It rained today but not yesterday.
- State 2: It rained yesterday but not today.
- State 3: It did not rain either yesterday or today.

The transition probability matrix P is given by:

$$P = \begin{pmatrix} 0{,}7 & 0 & 0{,}3 & 0 \\ 0{,}5 & 0 & 0{,}5 & 0 \\ 0 & 0{,}4 & 0 & 0{,}6 \\ 0 & 0{,}2 & 0 & 0{,}8 \end{pmatrix}$$

Given that it rained on Monday and Tuesday, we have the initial condition as state 0. To find the probability that it will rain on Thursday, we need to compute the two-step transition probabilities by squaring the matrix P:

$$P^{(2)} = P^2 = \begin{pmatrix} 0{,}7 & 0 & 0{,}3 & 0 \\ 0{,}5 & 0 & 0{,}5 & 0 \\ 0 & 0{,}4 & 0 & 0{,}6 \\ 0 & 0{,}2 & 0 & 0{,}8 \end{pmatrix}^2$$

After computing the matrix multiplication, we get:

$$P^{(2)} = \begin{pmatrix} 0{,}49 & 0{,}12 & 0{,}21 & 0{,}18 \\ 0{,}35 & 0{,}20 & 0{,}15 & 0{,}30 \\ 0{,}20 & 0{,}12 & 0{,}20 & 0{,}48 \\ 0{,}10 & 0{,}16 & 0{,}10 & 0{,}64 \end{pmatrix}$$

To calculate the probability of rain on Thursday, we sum the probabilities of being in state 0 or 1 on Thursday:

$$P_{00}^{(2)} + P_{01}^{(2)} = 0{,}49 + 0{,}12 = 0{,}61$$

Thus, the probability that it will rain on Thursday, given that it rained on Monday and Tuesday, is 0.61.

2.3 Characteristics of Markov Chains

1. **Transition Probabilities**: The probabilities of moving from one state to another are called transition probabilities. They are typically represented in a matrix called the transition matrix.

2. **State Space**: The set of all possible states that the chain can be in. This could be a finite or countably infinite set.

3. **Time Structure**: Transitions occur at integer time steps.

Example (Financial Market Dynamics). In financial market modeling, a Markov Chain captures the inherent volatility and unpredictability of the market. Here, the future state only depends on the current state, disregarding the historical path.
Market States:

- **Bull Market (Bull):** Optimistic phase with rising or expected-to-rise prices.
- **Bear Market (Bear):** Pessimistic phase with prolonged price declines.
- **Stagnant Market (Stagnant):** No clear trend; prices fluctuate within a narrow range.

Transition Matrix P:

$$P = \begin{pmatrix} 0{,}5 & 0{,}3 & 0{,}2 \\ 0{,}4 & 0{,}1 & 0{,}5 \\ 0{,}1 & 0{,}7 & 0{,}2 \end{pmatrix}$$

Two-Step Transition P^2:

$$P^2 = \begin{pmatrix} 0{,}39 & 0{,}32 & 0{,}29 \\ 0{,}29 & 0{,}48 & 0{,}23 \\ 0{,}35 & 0{,}24 & 0{,}41 \end{pmatrix}$$

Three-Step Transition P^3:

$$P^3 = \begin{pmatrix} 0{,}352 & 0{,}352 & 0{,}296 \\ 0{,}360 & 0{,}296 & 0{,}344 \\ 0{,}312 & 0{,}416 & 0{,}272 \end{pmatrix}$$

Interpretation: Transition matrices P^2 and P^3 offer insights into the market's longer-term dynamics. For instance, a Bull to Bear transition becomes more probable in three steps, highlighting the dynamic, unpredictable nature of financial markets modeled through Markov Chains.

Let $P_{ij}^{(n)}$ be the probability that the system transitions from state i to state j in n steps. Then:

$$P_{ij}^{(n)} = P(S_{t+n} = j | S_t = i) = (P^n)_{ij}$$

Notes:

- $P_{ij}^{(n)}$ can be found using the (i, j)th element of the matrix P^n.
- The potential paths from i to j in n steps are up to m^{n-1} (m being the number of states).
- Matrix multiplication of P by itself n times accumulates all transition probabilities.

Another representation of this concept, considering any times $s < t < u$, is given by:

$$P_{ij}(s, u) = \sum_{k \in S} P_{ik}(s, t) \cdot P_{kj}(t, u) \tag{2.3}$$

Consider the previously discussed financial market model. If we're interested in the probability of transitioning from a Bull market to a Bear market over five steps, P^5, we can employ the Chapman-Kolmogorov equation. Using our already computed matrices P^2 and P^3, the equation becomes:

$$(P^5)_{\text{Bull, Bear}} = \sum_k (P^2)_{\text{Bull, k}} \cdot (P^3)_{\text{k,Bear}} \tag{2.4}$$

Using the above equation, we find that:

$$(P^5)_{\text{Bull, Bear}} \approx 0{,}3526$$

Here, k represents all possible market states (Bull, Bear, Stagnant). This methodology not only simplifies computations but also offers insights into multi-step transitions in financial markets, enabling better predictive models.

Example. Consider a pensioner who receives 2 (thousand francs) at the beginning of each month. The amount required for monthly expenses is independent of his current capital and equals i with probability P_i, for $i = 1, 2, 3, 4$, ensuring $\sum_{i=1}^{4} P_i = 1$. Should the pensioner's end-of-month capital exceed 3, the excess is given to his son. Assuming an initial capital of 5, we investigate the probability of the pensioner's capital dropping to 1 or less within the next four months.
Solution: We model the pensioner's financial status as a Markov chain, with the state representing the end-of-month capital. To focus on scenarios where the capital drops to 1 or less, state 1 signifies such occurrences. Given the pensioner's practice of giving away excess capital, our analysis is limited to states 1, 2, and 3. The transition probability matrix $Q = [Q_{i,j}]$ is defined as:

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ P_3 + P_4 & P_2 & P_1 \\ P_4 & P_3 & P_1 + P_2 \end{pmatrix}.$$

Considering $P_i = 1/4$ for $i = 1, 2, 3, 4$, the matrix simplifies to:

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}.$$

Upon squaring this matrix twice, we obtain:

$$Q^{(4)} = \begin{pmatrix} 1 & 0 & 0 \\ 222/256 & 13/256 & 21/256 \\ 201/256 & 21/256 & 34/256 \end{pmatrix}.$$

Given the pensioner's starting capital as 3, the probability of it reducing to 1 or less within four months is $Q_{3,1}^{(4)} = 201/256$.

2.4 Limit Distribution

Limiting distribution, in the context of Markov chains, refers to the distribution to which the state probabilities converge as the number of steps (or time) goes to infinity.
Markov Chains, with their inherent ability to model complex stochastic systems, have significant applications across various domains, from finance and meteorology to social sciences. One of the most pivotal inquiries in the realm of Markov Chains pertains to their long-term behavior. Specifically, will the system stabilize into a steady state or equilibrium? This section delves into the notions that underpin this behavior, including the limit distribution and the properties of irreducibility, aperiodicity, and ergodicity.

Definition 2.3 (Limit Distribution). A distribution π is called a limit distribution for a Markov Chain if

$$\lim_{n\to\infty} P_{ij}^{n} = \pi_j$$

for every state i. The limit distribution provides insights into the enduring behavior of the system and is inherently linked to the properties discussed below.

The distribution π encapsulates the stable or steady-state probabilities associated with each state as the number of transitions grows indefinitely large. For a finite Markov Chain, the cumulative sum of all elements of π equals 1, emphasizing that π is a probability distribution.

Example (Convergence of P to π). For the given transition matrix P, let's inspect its powers:
For P^5:

$$\begin{bmatrix} 0{,}34296 & 0{,}35264 & 0{,}3044 \\ 0{,}34664 & 0{,}33984 & 0{,}31352 \\ 0{,}33752 & 0{,}3648 & 0{,}29768 \end{bmatrix}$$

For P^{10}:

$$\begin{bmatrix} 0{,}34260 & 0{,}35183 & 0{,}30557 \\ 0{,}34251 & 0{,}35210 & 0{,}30539 \\ 0{,}34268 & 0{,}35159 & 0{,}30573 \end{bmatrix}$$

For P^{20}:

$$\begin{bmatrix} 0{,}34259 & 0{,}35185 & 0{,}30556 \\ 0{,}34259 & 0{,}35185 & 0{,}30556 \\ 0{,}34259 & 0{,}35185 & 0{,}30556 \end{bmatrix}$$

By the time we examine P^{50} and P^{100}, the matrix has stabilized to:

$$\begin{bmatrix} 0{,}34259 & 0{,}35185 & 0{,}30556 \\ 0{,}34259 & 0{,}35185 & 0{,}30556 \\ 0{,}34259 & 0{,}35185 & 0{,}30556 \end{bmatrix}$$

From the matrices above, we discern a clear trend: as we raise T to higher powers, the rows of the matrix are converging to the limit distribution π. This showcases the theoretical underpinning that, given certain conditions, the Markov Chain will stabilize to a unique long-term distribution.

Definition 2.4. A Markov Chain is *irreducible* if it is possible to traverse from any state to any other state within a finite number of steps. Formally, for any states $i, j \in S$, there exists $n \geq 1$ such that $P_{ij}^{(n)} > 0$.

Irreducibility plays a paramount role in systems like social networks, ensuring the flow of information across the entire network.

Example. Consider a Markov chain with state space $\{1, 2, 3\}$ and transition matrix

$$P = \begin{pmatrix} 0{,}5 & 0{,}5 & 0 \\ 0 & 0{,}5 & 0{,}5 \\ 0{,}5 & 0 & 0{,}5 \end{pmatrix}.$$

This chain is irreducible because it is possible to move between any two states in at most 2 steps.

Definition The **period** of a state in a Markov chain is the greatest common divisor of all the lengths of paths that lead from the state back to itself. The period of state i is defined as $d(i) = \gcd\{n > 0 : P_{ii}^{n} > 0\}$.

Example. Consider a Markov chain with four states, A, B, C, and D, and the transition matrix P given by:

$$P = \begin{pmatrix} 0 & 0{,}5 & 0{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0{,}5 \\ 0{,}5 & 0 & 0 & 0{,}5 \\ 0 & 0{,}5 & 0{,}5 & 0 \end{pmatrix}.$$

The powers of the transition matrix P are calculated as follows:
For P^2 (the matrix squared):

$$P^2 = \begin{pmatrix} 0{,}5 & 0 & 0 & 0{,}5 \\ 0 & 0{,}5 & 0{,}5 & 0 \\ 0 & 0{,}5 & 0{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0{,}5 \end{pmatrix},$$

For P^3 (the matrix cubed):

$$P^3 = \begin{pmatrix} 0 & 0{,}5 & 0{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0{,}5 \\ 0{,}5 & 0 & 0 & 0{,}5 \\ 0 & 0{,}5 & 0{,}5 & 0 \end{pmatrix},$$

For P^4 (the matrix to the fourth power):

$$P^4 = \begin{pmatrix} 0{,}5 & 0 & 0 & 0{,}5 \\ 0 & 0{,}5 & 0{,}5 & 0 \\ 0 & 0{,}5 & 0{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0{,}5 \end{pmatrix}.$$

These results indicate the probabilities of transitioning from one state to another after 2, 3, and 4 steps, respectively. The cyclic pattern, where P^2 and P^4 are identical, and similarly P and P^3 are identical, suggests a periodicity in the Markov chain. Specifically, this implies that the system exhibits a period of 2, as the transition probabilities return to their original configuration every 2 steps. Thus, each state in this Markov chain has a period of 2, meaning it is possible to return to the same state in multiples of 2 steps.

Definition 2.5. A state of a Markov Chain exhibits *aperiodicity* if it doesn't revisit itself in a fixed pattern. Formally, a state i is aperiodic if the greatest common divisor of the set of steps n at which it returns to itself is one: $\gcd\{n : P_{ii}^{(n)} > 0\} = 1$.

Aperiodicity is crucial in financial models to avoid deterministic cyclical behaviors, ensuring the model captures the nuances of real-world dynamics.

Definition 2.6. A Markov Chain is termed *ergodic* if it embodies both irreducibility and aperiodicity. Ergodicity ensures the existence of a unique limit distribution π.

Theorem (Convergence to a Limiting Distribution): Let $\{X_n, n \geq 0\}$ be an irreducible, aperiodic Markov chain with a finite or countably infinite state space S. If π is a stationary distribution for this chain, then for any initial state $i \in S$,

$$\lim_{n\to\infty} P_{ij}^{(n)} = \pi(j),$$

where $P_{ij}^{(n)}$ represents the n-step transition probability from state i to state j, and the stationary distribution π satisfies

$$\pi(j) = \sum_{i\in S} \pi(i)P_{ij},$$

for all $j \in S$, ensuring $\sum_{j\in S} \pi(j) = 1$.
This theorem illustrates that under conditions of irreducibility and aperiodicity, the distribution of states of the Markov chain converges to a stationary distribution π, which is independent of the initial state. The stationary distribution π is characterized by the property that the long-term behavior of the chain can be described as a weighted sum of its immediate one-step transitions, governed by the transition probabilities P_{ij}.

Example. Two-State Markov Chain
Consider a Markov chain with two states, 1 and 2, and transition matrix

$$\mathsf{P} = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}5 & 0{,}5 \end{bmatrix}.$$

Objective: Find the limiting distribution.

Solution:
The stationary distribution π satisfies $\pi\mathsf{P} = \pi$ and $\pi_1 + \pi_2 = 1$. We solve

$$\begin{aligned} 0{,}9\pi_1 + 0{,}5\pi_2 &= \pi_1, \\ 0{,}1\pi_1 + 0{,}5\pi_2 &= \pi_2. \end{aligned}$$

This leads to $\pi_1 = \frac{5}{6}$ and $\pi_2 = \frac{1}{6}$.
Interpretation: The chain converges to a distribution where it is in state 1 with probability $\frac{5}{6}$ and in state 2 with probability $\frac{1}{6}$, regardless of the initial state.
Weather Model
Consider a weather model with states "Sunny"(S) and Rainy"(R), and transition matrix

$$\mathsf{P} = \begin{bmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{bmatrix}.$$

Objective: Determine the long-term weather forecast.
Solution: Solving for π in $\pi\mathsf{P} = \pi$ with $\pi_S + \pi_R = 1$ yields $\pi_S = 0{,}6$ and $\pi_R = 0{,}4$.

Long term, the forecast is sunny 60 % of the time and rainy 40 % of the time, regardless of initial weather.

Example. Three-State Markov Chain
Consider a Markov chain with states 1, 2, and 3, and transition matrix

$$\mathsf{P} = \begin{bmatrix} 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}1 & 0{,}6 & 0{,}3 \\ 0{,}4 & 0{,}1 & 0{,}5 \end{bmatrix}.$$

Objective: Find the limiting distribution.

Solution: We find the stationary distribution $\pi = [\pi_1, \pi_2, \pi_3]$ by solving $\pi\mathsf{P} = \pi$ subject to $\pi_1 + \pi_2 + \pi_3 = 1$. After solving, we interpret the probabilities of each state in the long term.

Example. Absorbing Markov Chain
Consider an absorbing Markov chain with states A (absorbing), 1, and 2, and transition matrix

$$\mathsf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0{,}1 & 0{,}8 & 0{,}1 \\ 0{,}2 & 0{,}2 & 0{,}6 \end{bmatrix}.$$

Objective: Probability of absorption starting from 1 and 2.

Solution: Calculate the fundamental matrix to find the absorption probabilities, leading to a clear understanding of long-term behavior towards the absorbing state.

Example. The Gambler's Problem is a classic scenario in stochastic processes, illustrating decision-making under uncertainty. A gambler has the opportunity to bet on the outcomes of a series of coin flips. If the coin comes up heads, the gambler wins as much money as they have bet; if it comes up tails, they lose their bet. The goal is to reach a certain amount of money, G, starting with an initial stake s. The game ends when the gambler reaches G or loses everything.
Objective: Determine the probability of reaching the goal G before going broke, given the initial stake s and the probability p of winning each bet.

Solution:
Let $P(s)$ denote the probability of reaching the goal starting with s. The probabilities satisfy the following recursive relationship:

$$P(s) = pP(s+1) + (1-p)P(s-1), \quad 0 < s < G,$$

with boundary conditions $P(0) = 0$ and $P(G) = 1$.
This difference equation can be solved using methods for linear homogeneous recurrence relations with boundary conditions.

Consider a simple case where $p = 0{,}5$ (a fair coin) and the goal $G = 4$. We seek the probability of reaching 4 starting from $s = 1$.
By solving the recurrence relation with the given boundary conditions, we find:

$$P(s) = \frac{s}{G},$$

for a fair game ($p = 0{,}5$). Therefore, the probability of reaching the goal of 4 starting with 1 is $P(1) = \frac{1}{4}$.
The Gambler's Problem showcases the use of stochastic models to evaluate probabilities of achieving certain states within a system governed by random processes. It highlights how initial conditions, transition probabilities, and boundary conditions play a crucial role in determining the outcome's likelihood.

Random Networks

3.1 Networks

Definition 3.1. A **Network** (or Graph) is a collection of entities called **Nodes** (or Vertices) and the relationships or connections between them, termed as **Edges** (or Links). Each edge connects two nodes and indicates a relationship between them.

We define the set of nodes V and the set of edges E as follows:

$$V = \{1, 2, 3, 4, 5\} \tag{3.1}$$

$$E = \{(1, 2), (2, 3), (1, 3), (2, 4), (3, 4), (4, 5)\} \tag{3.2}$$

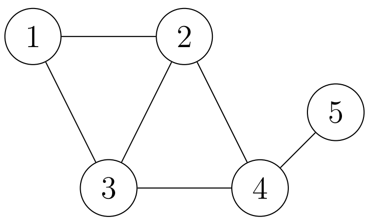


Figura 3.1: Example network with 5 nodes and 6 edges.

After defining networks in terms of nodes and edges, a mathematical representation is essential for analysis. The adjacency matrix offers a compact way to depict the relationships within a network.

Definition 3.2. The **Adjacency Matrix** A is a square matrix of size $N \times N$ where N is the total number of nodes. Entry A_{ij} equals 1 if there's an edge from node i to node j and 0 otherwise.

The adjacency matrix for the network in the previous example is:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

An adjacency matrix is a powerful tool for representing graphs because it encodes all the information about connections between nodes in a systematic manner. The cell A_{ij} represents the edge from node i to node j. If $A_{ij} = 1$, then an edge exists; otherwise, it's zero. This binary encoding simplifies complex relationships into a format easily analyzed computationally.

Definition 3.3. A **Shortest Path** between two nodes u and v in a graph is a path that has the minimum number of edges (in an unweighted graph) or the minimum sum of edge weights (in a weighted graph) among all possible paths between u and v. The length of this path is denoted as $d(u, v)$.

Definition 3.4. The **Diameter** of a graph is the longest shortest path between any two nodes in the graph. Formally, if $d(u, v)$ is the shortest path between nodes u and v, then the diameter D is defined as:

$$D = \max_{u,v \in V} d(u, v)$$

Let's consider the previous network example, where $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (2, 3), (1, 3), (2, 4), (3,$
The shortest paths between all pairs of nodes are:

- Shortest path from 1 to 2, 3, 4, 5 are $1, 1, 2, 3$ respectively.
- Shortest path from 2 to 1, 3, 4, 5 are $1, 1, 1, 2$ respectively.
- Shortest path from 3 to 1, 2, 4, 5 are $1, 1, 1, 2$ respectively.
- Shortest path from 4 to 1, 2, 3, 5 are $2, 1, 1, 1$ respectively.
- Shortest path from 5 to 1, 2, 3, 4 are $3, 2, 2, 1$ respectively.

The diameter of this graph is the longest of these shortest paths, which in this case is 3 (from node 1 to node 5).
An interesting property of adjacency matrices is that they can be multiplied to find paths of varying lengths between nodes. Specifically, A^2 (the matrix A multiplied by itself) will give us all possible paths of length 2 between any two nodes i and j.
For example, the element $(A^2)_{ij}$ will represent the number of paths of length 2 between nodes i and j. The logic behind this comes from the nature of matrix multiplication, where each element in the resulting matrix is computed as a sum of products involving elements from the corresponding row and column of the original matrices. In the context of networks, this translates to summing up possible intermediary steps to form paths of the length in question.
Given the adjacency matrix A from our previous example:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

We can calculate A^2 by multiplying A by itself. The resulting matrix A^2 will represent the number of paths of length 2 between any two nodes i and j.

$$A^2 = \begin{pmatrix} 2 & 1 & 1 & 1 & 0 \\ 1 & 3 & 2 & 1 & 1 \\ 1 & 2 & 3 & 1 & 1 \\ 1 & 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

For instance, the element $(A^2)_{13} = 1$ tells us there is exactly 1 path of length 2 from node 1 to node 3. This path is $1 \to 2 \to 3$.
Similarly, the element $(A^2)_{24} = 1$ indicates there is one path of length 2 from node 2 to node 4. This path is $2 \to 3 \to 4$.
Knowing the shortest paths and diameters in a network has a wide range of practical applications. For instance, in social networks, the diameter can provide insights into how quickly information may spread across the network. In transportation networks, identifying the shortest paths is crucial for optimizing travel routes. Understanding these properties is integral for network resilience, efficiency, and information dissemination.
Before delving into the fascinating properties of random graphs, let's establish what it means for a set of nodes to be connected within a graph. This idea naturally extends from our discussion about paths of varying lengths.

Definition 3.5 (Connected Component)**.** A **Connected Component** in a network is a maximal set of nodes C such that for every pair of nodes a, b in C, there exists an undirected path from a to b.

To identify connected components, one may start at an arbitrary node and find all nodes reachable from it. This set forms one connected component, and the process is repeated until all nodes are included in a connected component.
Consider our earlier example network with adjacency matrix A as follows:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

We already calculated A^2 which provides us with all paths of length 2 between any two nodes i and j:

$$A^2 = \begin{pmatrix} 2 & 1 & 1 & 2 & 0 \\ 1 & 3 & 2 & 1 & 1 \\ 1 & 2 & 3 & 1 & 1 \\ 2 & 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Now, let's calculate A^3 to find all paths of length 3 between any two nodes:

$$A^3 = A \times A^2 = \begin{pmatrix} 0 & 3 & 3 & 1 & 1 \\ 3 & 0 & 3 & 3 & 1 \\ 3 & 3 & 0 & 3 & 1 \\ 1 & 3 & 3 & 0 & 2 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix}$$

For instance, the element $(A^3)_{14}$ is 1, which means there is one path of length 3 between node 1 and node 4. Similarly, $(A^3)_{52}$ is also 1, indicating a single path of length 3 between nodes 5 and 2.
These calculations reinforce the notion that the network consists of a single connected component $C = \{1, 2, 3, 4, 5\}$, as all nodes are reachable from one another via paths of varying lengths.

Having reviewed the basic notations and mathematical properties of networks, in the upcoming sections, we'll delve deeper into various network models. These models will help illuminate the structures and intricacies of networks we encounter daily, from social media connections to vast communication networks. With these tools, we aim to foster a richer understanding of how systems and patterns interconnect in the world around us.

3.2 Random Graphs

Random graphs are a foundational construct in the mathematical treatment of network theory, offering researchers a framework for understanding the probabilistic interactions within complex networks.

Definition 3.6 (Random Network)**.** A **Random Network** is a graph in which the presence or absence of an edge between any two distinct nodes is determined by a random process or probabilistic rule.

Among various random graph models, the Erdos-Renyi model stands out due to its simplicity and foundational role in network theory.

Definition 3.7 (Erdos-Renyi Model)**.** The **Erdos-Renyi Model**, denoted as $G(n, p)$, is defined as a random graph consisting of n nodes, where each potential edge between distinct nodes i and j is included with probability p, independently of the other edges.

The probability of an edge forming between any two nodes i and j in the Erdos-Renyi model $G(n, p)$ is:

$$P(\text{Edge between } i \text{ and } j) = p \tag{3.3}$$

Definition 3.8 (Giant Connected Component)**.** A **Giant Connected Component** in a graph is a connected component that includes a substantial fraction of the entire set of nodes in the graph. In the Erdos-Renyi model, a giant connected component emerges when the edge probability p surpasses a critical value p_c.

The critical probability p_c for the emergence of a giant component in an Erdos-Renyi graph $G(n, p)$ is approximately:

$$p_c \approx \frac{\log(n)}{n} \tag{3.4}$$

[Empirical Estimation of p_c] To empirically estimate the critical probability p_c at which a giant connected component forms in an Erdos-Renyi graph $G(n, p)$, we will perform the following experiment:

1. Initialize $n = 1000$ nodes.

2. Vary p from 0 to 1 in increments of 0.01.

3. For each p, generate a random graph $G(n, p)$.
4. Identify the largest connected component C in G.
5. Record the size $|C|$ of the largest connected component.
6. Plot $|C|$ as a function of p.
7. Observe the value of p where $|C|$ starts to dramatically increase. This value is an empirical estimate of p_c.

The Erdos-Renyi model, despite its simplistic assumptions, serves as a key reference model in the realm of network theory. Its clear framework for randomness offers a foundation for the study of more complex networks, making it a cornerstone in disciplines such as computer science, biology, and social sciences.
The Preferential Attachment Model, popularized by Albert-László Barabási and Réka Albert, is grounded in the adage ”the rich get richer.”Nodes are more likely to link to nodes that already have many connections. This dynamic leads to ”hubs.^{o}r nodes with significantly higher connectivity than others.
Mathematically, the probability $\Pi(k)$ that a new node connects to a node with k connections is proportional to k. This results in a scale-free network, characterized by a power-law degree distribution.
The Preferential Attachment Model is a foundational concept often used to explain how real-world networks evolve to exhibit a scale-free degree distribution. Proposed by Barabási and Albert in 1999, this model argues that networks grow by the principle of ”the rich get richer,”whereby new nodes are more likely to connect to already well-connected nodes.
The central equation governing the Preferential Attachment Model is:

$$\Pi(k) = \frac{k}{\sum_j k_j} \tag{3.5}$$

where $\Pi(k)$ is the probability that a new node will connect to a node with degree k, and the sum runs over all nodes j in the network.
The Preferential Attachment Model provides a basis for understanding how highly connected ”hubs.^{em}erge in networks. These hubs play a critical role in the network's overall structure and resilience, often dominating processes like information spread or failure propagation.
The Preferential Attachment Model finds applications in various fields, including the World Wide Web, citation networks, and even biological systems, to explain phenomena like protein-protein interaction networks.

3.3 Branching Processes

Branching processes are stochastic models that describe the dynamics of populations where individuals reproduce independently. These models are widely used in biology for understanding population dynamics and disease spread, as well as in physics for particle decay simulations.

Definition 3.9 (Branching Process). A branching process $\{Z_n\}_{n=0}^{\infty}$ is defined where Z_n denotes the number of individuals in the n-th generation. Initiated with $Z_0 = 1$, the process evolves according to:

$$Z_{n+1} = \sum_{i=1}^{Z_n} X_{n,i},$$

where $\{X_{n,i}\}$ are i.i.d. random variables representing the offspring of the i-th individual in generation n.

The Galton-Watson process is a notable example of a branching process, used to study extinction probabilities and the effects of varying reproductive rates.

Definition 3.10 (Galton-Watson Process)**.** A Galton-Watson process involves each individual in the population reproducing independently according to a fixed probability distribution, starting from a single ancestor.

Example. Consider a Galton-Watson process where each individual reproduces with either 0 or 2 offspring, each outcome equally likely. This model explores outcomes like potential extinction if no offspring are produced, or exponential growth if reproduction rates are maximized.

The expected number of offspring per individual, μ, is given by:

$$\mu = \sum_{j=0}^{\infty} jP_j,$$

and the variance in the number of offspring, σ^2, is:

$$\sigma^2 = \sum_{j=0}^{\infty} (j-\mu)^2 P_j.$$

The probability of eventual extinction, π_0, for a population is crucial for understanding its long-term sustainability and is defined by:

$$\pi_0 = \sum_{j=0}^{\infty} P_j(\pi_0)^j.$$

Example. If a branching process has an offspring distribution with $P_0 = 0{,}5$ and $P_2 = 0{,}5$, then the mean $\mu = 1$ and variance $\sigma^2 = 0{,}5$, illustrating a critical threshold where the population's fate hinges on initial fluctuations.

These elements highlight how branching processes model complex population dynamics, emphasizing the interplay between stochasticity in reproduction and the overarching trends in population growth or decline.

3.4 Time Reversible Markov Chain

Time Reversible Markov Chains facilitate analysis of Markov processes in reverse time while preserving Markovian characteristics. For a Markov chain with:

- Transition probabilities: P_{ij}, transitioning from state i to state j.
- Stationary distribution: π_i, the long-term state probabilities.

The reversed chain's transition probabilities, Q_{ij}, are calculated as:

$$Q_{ij} = \frac{\pi_j P_{ji}}{\pi_i} \tag{3.6}$$

For time reversibility, the detailed balance condition must be satisfied:

$$\pi_i P_{ij} = \pi_j P_{ji}, \quad \forall i, j \tag{3.7}$$

This condition ensures the forward and reverse processes are indistinguishable at equilibrium, reflecting a symmetry in transitions.

Implications of Time Reversibility

Time reversibility imposes a unique equilibrium structure on the Markov chain, leading to:

1. **Symmetric Behavior:** The chain exhibits a balance in transitions, allowing forward and backward analysis.
2. **Ergodicity:** For ergodic chains, time reversibility ensures uniform mixing over time, strengthening the chain's stochastic properties.
3. **Inference and Estimation:** Time reversibility simplifies the estimation of transition probabilities and stationary distributions, facilitating easier model parameterization.

Monte Carlo Markov Chain (MCMC)

Monte Carlo Markov Chain (MCMC) methods are essential for sampling from complex probability distributions by constructing a Markov chain that has the desired distribution as its stationary distribution. The main objective is to utilize these samples for approximating integrals, optimizing functions, and exploring properties of distributions that are difficult to analyze analytically.
An MCMC method constructs a Markov chain such that for every pair of states x and y, the detailed balance condition with respect to π is satisfied:

$$\pi_x P_{xy} = \pi_y P_{yx}, \quad \forall x, y \tag{3.8}$$

Metropolis-Hastings Algorithm:

1. Begin with an initial state x_0.
2. For each iteration $t = 1, 2, \ldots, T$:

a) Propose a new state y from a proposal distribution $q(y|x_{t-1})$.

b) Calculate the acceptance ratio $a = \frac{\pi_y q(x_{t-1}|y)}{\pi_{x_{t-1}} q(y|x_{t-1})}$.

c) Accept the new state with probability $\min(1, a)$, resulting in $x_t = y$; otherwise, retain $x_t = x_{t-1}$.

Gibbs Sampling: This algorithm is particularly useful for sampling from multivariate distributions. It sequentially updates each component of the state vector, sampling from the conditional distribution of each component given all other components, thus facilitating efficient exploration of the state space.

Illustration: Sampling from a Biased Coin Distribution

To demonstrate the application of MCMC, consider sampling from the distribution of a biased coin's outcomes, where the probability of heads (H) is $\pi_H = 0{,}7$ and tails (T) is $\pi_T = 0{,}3$.
Procedure:

1. Define the target distribution $\pi(x)$ with $\pi_H = 0{,}7$ and $\pi_T = 0{,}3$.

2. Employ a proposal distribution $q(y|x)$ that allows transitions between states with a certain probability.

3. Use the Metropolis-Hastings algorithm to accept or reject proposed transitions, thereby generating a sample from the target distribution.

Proof of Convergence for the Metropolis-Hastings Algorithm

To demonstrate that the Markov chain generated by the Metropolis-Hastings algorithm converges to the desired stationary distribution π, we rely on the detailed balance condition. This condition is essential for ensuring that the Markov chain is time-reversible and reaches equilibrium.

Detailed Balance Condition

The detailed balance condition for a Markov chain with stationary distribution π is given by:

$$\pi_i P_{ij} = \pi_j P_{ji}, \quad \forall i, j \tag{3.9}$$

Metropolis-Hastings Algorithm Steps

The Metropolis-Hastings algorithm involves two steps: proposal and acceptance-rejection.

1. **Proposal:** Propose a transition from the current state i to a new state j using a proposal distribution q_{ij}.

2. **Acceptance-Rejection:** The acceptance ratio a_{ij} for transitioning from state i to state j is computed as:

$$a_{ij} = \frac{\pi_j q_{ji}}{\pi_i q_{ij}} \tag{3.10}$$

The transition is accepted with probability $\alpha_{ij} = \text{mín}(1, a_{ij})$, resulting in the effective transition probability:

$$P_{ij} = q_{ij}\alpha_{ij} \tag{3.11}$$

For non-accepted transitions, the chain remains in the current state, ensuring the probabilities sum to 1.

Proof of Convergence Using Detailed Balance

To verify that the detailed balance condition is satisfied and thus prove convergence to the distribution π, we examine the effective transition probabilities:

$$\begin{aligned}\pi_i P_{ij} &= \pi_i q_{ij}\alpha_{ij}\\ &= \pi_i q_{ij}\,\text{mín}\left(1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right)\end{aligned}$$

For the reverse transition from state j to state i, the symmetry in the mín function allows us to write:

$$\begin{aligned}\pi_j P_{ji} &= \pi_j q_{ji}\alpha_{ji}\\ &= \pi_j q_{ji}\,\text{mín}\left(1, \frac{\pi_i q_{ij}}{\pi_j q_{ji}}\right)\end{aligned}$$

By design, the Metropolis-Hastings algorithm's acceptance-rejection step ensures that for all state pairs (i, j), the detailed balance condition:

$$\pi_i P_{ij} = \pi_j P_{ji} \tag{3.12}$$

is satisfied, proving that π is indeed the stationary distribution of the Markov chain generated by the algorithm.

3.5 Markov Decision Processes (MDP)

A Markov Decision Process (MDP) provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. MDPs are useful in studying optimization problems solved via dynamic programming and reinforcement learning.
An MDP is defined as a tuple (S, A, P, R) where:

- S is a finite set of states.
- A is a finite set of actions.
- P is the state transition probability function, $P_{ij}(a)$, representing the probability of transitioning from state i to state j under action a.

- R is the reward function, $R(i, a)$ or $R(i, j, a)$, specifying the reward received when transitioning from state i to state j due to action a.

The goal within an MDP framework is to discover a policy π that maximizes the expected cumulative reward from any initial state over a horizon, which can be finite or infinite.
A policy π specifies the action a to be taken in each state s. The optimal policy π^* is the one that maximizes the expected cumulative reward over time, which can be found using dynamic programming techniques, such as value iteration or policy iteration, or through direct policy search methods in reinforcement learning contexts.
The challenge in solving MDPs lies in the trade-off between immediate rewards and long-term gains, requiring careful consideration of future state probabilities and rewards when choosing actions.

Definition 3.11 (Markov Decision Process). A Markov Decision Process is defined by a tuple (S, A, P, R), where S is the set of states, A is the set of actions, P is the state transition probability function $P : S \times A \times S \to [0, 1]$, and R is the reward function $R : S \times A \to \mathbb{R}$.

A policy β specifies the action to be chosen in each state, potentially as a probability distribution over actions. The goal is to identify a policy that maximizes the expected average reward over time. For any policy β, the expected reward when taking action a in state i is given by $R(i, a)$. The steady-state probability of being in state i and taking action a under policy β is denoted by π_{ia}, satisfying:

$$\sum_{a \in A} \pi_{ia} = 1, \quad \forall i \in S \tag{3.13}$$

$$\pi_{ja} = \sum_{i \in S} \pi_{ia} P_{ij}(a), \quad \forall j \in S, a \in A \tag{3.14}$$

The expected average reward under policy β is then given by:

$$\text{Expected Average Reward}(\beta) = \sum_{i \in S} \sum_{a \in A} \pi_{ia} R(i, a) \tag{3.15}$$

The aim is to find a policy β that maximizes this expected average reward.
Grid Navigation MDP: Consider a grid where an agent aims to move from a start position to a goal position with minimal steps. The states S represent grid cells, and the actions $A = \{up, down, left, right\}$ move the agent between states. Assume all movements have a reward of -1, encouraging the shortest path, and reaching the goal yields a reward of $+10$.
If the agent attempts to move into a wall or outside the grid, it remains in its current state. Let's consider a simplified 2x2 grid with states $\{1, 2, 3, 4\}$, where state 4 is the goal, and state 3 is an obstacle. The transition probabilities $P_{ij}(a)$ for moving from state i to state j with action a might look like:

$$P_{11}(up) = 0, \qquad P_{11}(right) = 1, \qquad \text{and so on.}$$

The reward function $R(i, a)$ for moving from state i using action a:

$$R(1, \mathrm{right}) = -1, \qquad R(2, \mathrm{right}) = +10.$$

To determine the optimal policy, one could use value iteration to calculate the value of each state and derive the policy that maximizes rewards.
Gambling MDP: A gambler with an initial wealth of W dollars can bet any integer amount $b \leq W$ on a fair coin toss. Winning doubles the bet, while losing forfeits it. Here, states S represent the gambler's current wealth, and actions A are the possible bet sizes. The goal is to reach a wealth of W_{max}.
The state transition probabilities for betting b dollars are:

$$P_{W,W+b}(b) = 0{,}5, \qquad P_{W,W-b}(b) = 0{,}5.$$

The reward function $R(W, b)$ could be defined as:

$$R(W, b) = \begin{cases} +1 & \text{if } W + b = W_{max} \\ -1 & \text{if } W - b = 0 \\ 0 & \text{otherwise} \end{cases}$$

Poisson Process

4.1 Counting processes

Poisson processes are a fundamental concept in stochastic modeling, providing a rigorous mathematical framework for understanding events that occur randomly in time or space. Arrival times and counting processes such as Poisson processes find applications in a myriad of contexts, each with its own set of challenges and implications. For instance, in healthcare, modeling the arrival times of patients in an emergency room can be crucial for optimizing resource allocation and improving patient outcomes. Similarly, understanding the time intervals between bus arrivals at a specific stop can offer insights into public transportation scheduling and efficiency. In the realm of computer science, the arrival times of data packets in a network can be analyzed to optimize bandwidth and reduce latency. Businesses too can benefit; for example, modeling the arrival times of customers in a service queue, whether in a call center or a fast-food restaurant, can lead to enhanced service management. Natural events like earthquakes, floods, and forest fires also exhibit arrival times that can be modeled to better understand and predict these phenomena. In retail, the time between customer arrivals at a checkout counter can inform decisions about staffing and service speed. Social media platforms often scrutinize the timing of posts or mentions to understand user engagement or to detect trending topics. In manufacturing, arrival times of components on an assembly line can be critical for identifying bottlenecks and optimizing production. Financial markets are another fertile ground where the arrival times of buy/sell orders can shed light on market dynamics. Finally, in ecology, monitoring the arrival times of different species at a watering hole or feeding station can offer invaluable data for conservation efforts and ecological research.

Definition 4.1 (Counting Process). A counting process is a stochastic process $\{N(t), t \geq 0\}$ that represents the total number of events that have occurred up to time t. The function $N(t)$ satisfies the following properties:

1. $N(0) = 0$ (initial condition)
2. $N(t)$ is integer-valued for all $t \geq 0$
3. $N(t)$ is non-decreasing as t increases; that is, if $s < t$, then $N(s) \leq N(t)$
4. The function $N(t)$ is right-continuous, meaning that for each t, $\text{lím}_{s\to t^+} N(s) = N(t)$

In simpler terms, a counting process counts the number of times a certain event has occurred by any given time t. The count starts at zero and can only increase as time moves forward.
Consider a time interval T that we divide into n smaller intervals, each of length $\Delta t = \frac{T}{n}$. We are interested in counting the number of occurrences of a particular event within each small time interval Δt.
Initially, let's model this as a Bernoulli process. In each small time interval Δt, the event can

either occur with probability p or not occur with probability $1-p$.

$$P(\text{Event occurs in } \Delta t) = p \tag{4.1}$$

$$P(\text{Event does not occur in } \Delta t) = 1-p \tag{4.2}$$

For large n and small Δt, we can relate p to a rate parameter λ as follows:

$$p = \lambda \Delta t \tag{4.3}$$

Now, let's consider the number of events X that occur in the entire interval T. The variable X is a sum of n independent Bernoulli random variables, each with success probability p. Therefore, X follows a binomial distribution:

$$X \sim \text{Binomial}(n,p) \tag{4.4}$$

The probability of observing exactly k events in T is given by:

$$P(X=k) = \binom{n}{k} p^k (1-p)^{(n-k)} \tag{4.5}$$

Substitute $p = \lambda\Delta t$ and $1-p = 1-\lambda\Delta t$:

$$P(X=k) = \binom{n}{k} (\lambda\Delta t)^k (1-\lambda\Delta t)^{(n-k)} \tag{4.6}$$

The binomial coefficient can be expanded as:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!} \tag{4.7}$$

Substitute this into the probability mass function:

$$P(X=k) = \frac{n(n-1)\cdots(n-k+1)}{k!} (\lambda\Delta t)^k (1-\lambda\Delta t)^{(n-k)} \tag{4.8}$$

The first limit becomes 1 as n gets larger and larger. The second limit turns into $\exp(-\lambda T)$ in the same way. These observations lead us to a key theorem about how the Bernoulli process evolves into a Poisson process.

Theorem 4.2 (Convergence from Bernoulli to Poisson)**.** Let X be the number of events in a time interval T broken down into n smaller intervals. Each smaller interval has length $\Delta t = \frac{T}{n}$. If each interval has a Bernoulli-distributed event occurrence with probability $p = \lambda\Delta t$, then as n approaches infinity with $n\Delta t = T$ constant, the distribution of X turns into a Poisson distribution. Specifically,

$$\lim_{n\to\infty} \binom{n}{k} p^k (1-p)^{(n-k)} = \frac{(\lambda T)^k}{k!} \exp(-\lambda T) \tag{4.9}$$

In this limit, X follows a Poisson distribution with parameter λT.

4.2 Poisson Processes

In the world of stochastic processes, the Poisson process holds a place of prominence for its mathematical elegance and wide-ranging applicability. It's a vital tool in various domains such as queuing theory, telecommunications, and even quantum physics. The Poisson process serves as a mathematical model for situations where events occur randomly in time or space.

4.2.1. Homogeneous Poisson Process

We start by introducing the most straightforward version of the Poisson process, the Homogeneous Poisson Process. In this variant, the rate at which events happen is constant over time, making it a natural extension of the Bernoulli process under limiting conditions.

Definition 4.3 (Homogeneous Poisson Process)**.** Let $(N(t) : t \geq 0)$ be a counting process. $N(t)$ is said to be a *Homogeneous Poisson Process* with rate $\lambda > 0$ if the following conditions hold:

1. $N(0) = 0$
2. The increments are independent.
3. The number of events in any interval of length t follows a Poisson distribution with mean λt.

The Homogeneous Poisson Process is uniquely characterized by its rate parameter λ, which tells us the average number of events per unit time. It's called 'homogeneous' because this rate is constant across time. This process provides a stochastic model for a variety of real-world phenomena where events occur continuously and independently at a constant average rate.

The concept of waiting times is crucial for understanding any stochastic process, and the Poisson process is no exception. In a Homogeneous Poisson Process, the waiting times between successive events are exponentially distributed.

Theorem 4.4 (Exponential Waiting Times)**.** In a Homogeneous Poisson Process with rate λ, the time T until the first event occurs follows an exponential distribution with parameter λ, i.e.,

$$P(T \leq t) = 1 - e^{-\lambda t}$$

This theorem can be derived from the properties of the Poisson process and provides essential insights into the behavior of the system. For example, it tells us that the process has no memory, meaning the time until the next event is independent of the past.

To prove that the waiting times are exponentially distributed in a Homogeneous Poisson Process, let's consider the probability that no event occurs in the interval $[0, t]$. According to the definition of a Homogeneous Poisson Process, the number of events $N(t)$ in any interval $[0, t]$ follows a Poisson distribution with mean λt. Therefore,

$$P(N(t) = 0) = \frac{e^{-\lambda t}(\lambda t)^0}{0!} = e^{-\lambda t} \tag{4.10}$$

Now, the time T until the first event occurs is greater than t if and only if no event occurs in the interval $[0, t]$. Therefore,

$$P(T > t) = P(N(t) = 0) = e^{-\lambda t} \tag{4.11}$$

To find the distribution of T, we can find its cumulative distribution function (CDF), which is given by $P(T \leq t)$. The CDF is the complement of $P(T > t)$:

$$P(T \leq t) = 1 - P(T > t) = 1 - e^{-\lambda t} \tag{4.12}$$

Differentiating both sides with respect to t gives us the probability density function (PDF) of T:

$$f_T(t) = \frac{d}{dt} P(T \leq t) = \lambda e^{-\lambda t} \tag{4.13}$$

This is the PDF of an exponential distribution with rate parameter λ, completing the proof.
The memoryless property is a unique feature of the exponential distribution that has significant implications for the Poisson process. In mathematical terms, the memoryless property for an exponentially distributed random variable T with rate λ is described as follows:

$$P(T > s + t \mid T > s) = P(T > t) \quad \text{for all } s, t \geq 0 \tag{4.14}$$

This equation states that the probability that we have to wait an additional t time units given that we've already waited s time units is the same as if we had not waited at all.
To prove the memoryless property, we need to show that the conditional probability $P(T > s + t \mid T > s)$ equals $P(T > t)$.
Starting with the definition of conditional probability:

$$\begin{aligned} P(T > s + t \mid T > s) &= \frac{P(T > s + t \,\text{and}\, T > s)}{P(T > s)} \\ &= \frac{P(T > s + t)}{P(T > s)} \end{aligned}$$

We've used the fact that $T > s + t$ implies $T > s$, which allows us to simplify the numerator.
Now, we know that T is exponentially distributed with rate λ, so:

$$\begin{aligned} P(T > s + t \mid T > s) &= \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} \\ &= e^{-\lambda t} \\ &= P(T > t) \end{aligned}$$

This completes the proof of the memoryless property.

Theorem 4.5 (Memoryless Property). The waiting times in a Homogeneous Poisson Process are memoryless, i.e., for any $s, t \geq 0$,

$$P(T > s + t \mid T > s) = P(T > t)$$

4.2.2. Non-Homogeneous Poisson Process

A Homogeneous Poisson Process assumes a constant rate of events, which may not adequately model dynamic real-world scenarios where the event rate fluctuates over time, such as in web traffic analysis or emergency room visit frequencies. The Non-Homogeneous Poisson Process (NHPP) generalizes this by incorporating a rate function, $\lambda(t)$, that varies with time.

A counting process $N(t)$ for $t \geq 0$ is defined as a Non-Homogeneous Poisson Process if it satisfies the following criteria:

1. $N(0) = 0$.
2. The increments $N(t) - N(s)$ for $0 \leq s < t$ are independent.
3. The number of events in any interval $[s, t]$ is Poisson distributed with mean given by $\int_s^t \lambda(u)\, du$, where $\lambda(u)$ is a time-dependent rate function.

The rate function $\lambda(t)$ should be non-negative for all t and may be specified in various forms (e.g., linear, sinusoidal, step function) to appropriately model different time-dependent behaviors.

Simulation

5.1 Monte-Carlo Simulation

Let $\mathbf{X} = (X_1, \ldots, X_n)$ denote a random vector having a given density function $f(x_1, \ldots, x_n)$ and suppose we are interested in computing

$$E[g(\mathbf{X})] = \int \cdots \int g(x_1, \ldots, x_n) f(x_1, \ldots, x_n)\, dx_1 \cdots dx_n$$

for some n-dimensional function g. For instance, g could represent the total delay in queue of the first $\lfloor n/2 \rfloor$ customers when the $\mathbf{X}$ values represent the first $\lfloor n/2 \rfloor$ inter-arrival and service times. In many situations, it is not analytically possible either to compute the preceding multiple integral exactly or even to numerically approximate it within a given accuracy. One possibility that remains is to approximate $E[g(\mathbf{X})]$ by means of simulation.
To approximate $E[g(\mathbf{X})]$, start by generating a random vector

$$\mathbf{X}^{(1)} = \begin{pmatrix} X_1^{(1)} \\ \vdots \\ X_n^{(1)} \end{pmatrix}$$

having the joint density $f(x_1, \ldots, x_n)$ and then compute

$$Y^{(1)} = g(\mathbf{X}^{(1)}).$$

Now generate a second random vector (independent of the first) $\mathbf{X}^{(2)}$ and compute $Y^{(2)} = g(\mathbf{X}^{(2)})$. Keep on doing this until r, a fixed number of independent and identically distributed random variables $Y^{(i)} = g(\mathbf{X}^{(i)})$, $i = 1, \ldots, r$ have been generated. Now by the strong law of large numbers, we know that

$$\lim_{r \to \infty} \frac{Y^{(1)} + \ldots + Y^{(r)}}{r} = E[Y^{(i)}] = E[g(\mathbf{X})]$$

and so we can use the average of the generated Ys as an estimate of $E[g(\mathbf{X})]$. This approach to estimating $E[g(\mathbf{X})]$ is called the *Monte Carlo simulation*.

Fundamental Theorem of Monte Carlo Integration

Building on the Law of Large Numbers, Monte Carlo Integration approximates integrals by averaging function values at randomly chosen points.

Theorem 5.1 (Monte Carlo Integration). Consider a real-valued function $f(x)$ defined over a domain D. The Monte Carlo estimate for the integral $\int_D f(x)\, dx$ is:

$$\int_D f(x)\, dx = \frac{1}{N} \sum_{i=1}^{N} f(x_i),$$

where x_i are random samples drawn uniformly from D. As N approaches infinity, and under certain conditions, this estimate converges to the true value of the integral, thanks to the Law of Large Numbers.

Monte Carlo Integration can be approached using indicator random variables, especially useful when the domain of integration, D, is complex or irregularly shaped. This method leverages random sampling and probability to provide an estimate for the integral.
The methodology is:

1. **Random Sampling**: Draw a random point (u_1, u_2) uniformly from a larger domain R that encompasses D.

2. **Indicator Variable**: Define a binary random variable I as:

$$I = \begin{cases} 1 & \text{if } u_1 \leq f(u_2) \text{ and } (u_1, u_2) \in D \\ 0 & \text{otherwise} \end{cases}$$

   This variable I is 1 if the point (u_1, u_2) lies below the curve of f within D, and 0 otherwise.

3. **Compute the Proportion**: After drawing N random points, compute the proportion $\hat{p}$ of points for which $I = 1$. This proportion estimates the ratio of the area under f in D to the area of R.

4. **Estimate the Integral**: The integral of f over D is approximately $\hat{p} \times |R|$.

The expectation of the indicator random variable I is given by:

$$\mathbb{E}[I] = P((u_1, u_2) \text{ is under } f \text{ and in } D)$$

This expectation is essentially the proportion of the area under f within D relative to R:

$$\mathbb{E}[I] = \frac{\text{Area under } f \text{ in } D}{|R|}$$

The Monte Carlo estimate for this expectation, after N trials, is:

$$\hat{p} = \frac{1}{N} \sum_{i=1}^{N} I_i$$

where I_i is the value of I for the i-th random sample. Thus, the Monte Carlo estimate for the integral of f over D becomes:

$$\hat{p} \times |R|$$

By the Law of Large Numbers, as N grows larger, $\hat{p}$ converges to $\mathbb{E}[I]$, making our integral estimate increasingly accurate.

Example (Estimation of π using Monte Carlo). Consider a unit circle inscribed in a unit square. If we uniformly sample random points within this square, the probability that a point lies inside the circle is equal to the ratio of the area of the circle to the area of the square. Given that the area of the unit circle is π and the area of the unit square is 1, this ratio is $\frac{\pi}{4}$.
Let's define an indicator random variable I:

$$I = \begin{cases} 1 & \text{if the point is inside the unit circle} \\ 0 & \text{otherwise} \end{cases}$$

After drawing N random points in the square, the proportion $\hat{p}$ of points for which $I = 1$ approximates the ratio of the area of the circle to the square. Therefore, an estimate of π is given by:

$$\pi \approx 4 \times \hat{p}$$

This method leverages the geometric interpretation of π and the probabilistic foundations of Monte Carlo to provide an estimate. As N grows larger, the estimate becomes more accurate due to the Law of Large Numbers.

Example. Consider the function $f(x) = x^2$ over the interval $[0, 1]$. The actual value of this integral is $\frac{1}{3}$. Using Monte Carlo integration, we can estimate this value.
As seen in the Figure, the blue curve represents the function $f(x) = x^2$. The green dots represent the random points that fall below the curve, and the red dots represent the points that fall above the curve. Through this method, we estimated the value of the integral to be approximately $0{,}3349$, which is close to the actual value of $\frac{1}{3} \approx 0{,}3333$.

5.2 Random Number Generators

Definition 5.2 (Random number). A random number is an unpredictable value, generated independently from preceding or succeeding numbers. It lacks any discernible pattern or regularity, making it impossible to deduce without understanding the underlying random generation process. Additionally, a random number should accurately represent true randomness, ensuring an equitable chance for all potential outcomes.

Moving from individual random numbers, it is essential to understand how we can generate a series of such numbers, which leads us to the concept of a random number generator.

Definition 5.3 (Random number Generator). A **Random Number Generator** (RNG) is an algorithm that produces a sequence of numbers that lacks any pattern, i.e., appears random.
More formally, an RNG is defined as a function:

$$\begin{aligned} R :& S \to T \\ & (s) \mapsto t \end{aligned}$$

where:

- S is the seed space, a finite set of initial states. An RNG is typically initialized with a value in S, known as the *seed*.
- T is the target space, typically the set of real numbers in the interval [0, 1) or a set of integer values.
- The function R maps each seed $s \in S$ to a target $t \in T$ in a manner that appears random.

RNGs are essential in many areas of computing, including simulation, cryptography, and probabilistic algorithms. While the outputs of an RNG may appear random, they are determined entirely by the initial seed and are thus *pseudorandom*.
To get closer to ”truerandomness in computer systems, one approach is to use some fundamentally unpredictable process as a source of randomness. These are known as hardware (or true) random number generators (HRNGs or TRNGs).
For example, they might use physical processes like atmospheric noise, radioactive decay, or even small variations in the timing of keyboard presses or mouse movements. These sources are inherently unpredictable and do not follow a deterministic algorithm, so the numbers generated in this way can be considered truly random.
However, HRNGs tend to be slower and more difficult to implement than PRNGs, and in many cases, the numbers generated by PRNGs are sufficiently random for the task at hand.
With the understanding of what random numbers and their generators are, we can now lay down some properties that a well-functioning random number generator should exhibit:

- **Unpredictability:** Without knowing the algorithm and seed, it should be impossible to predict future numbers.
- **Reproducibility:** Given the same seed, the RNG should produce the same sequence of numbers.
- **Representation of True Randomness:** The RNG should accurately represent true randomness, ensuring an equitable chance for all potential outcomes.
- **Long period:** The sequence of numbers should be long before repeating.
- **Efficiency:** The RNG should generate numbers quickly.

It’s important to note that not all random number generators will have all these properties. For instance, cryptographic random number generators prioritize unpredictability and may sacrifice reproducibility. The appropriate RNG for a given application depends on what properties are most important for that use case.
The Linear Congruential Generator (LCG) is a type of pseudorandom number generator, and it is one of the oldest and best-known pseudorandom number generator algorithms. The simplicity of its underlying mathematical structure, combined with its fast execution and the minimal memory it requires, have contributed to its widespread usage.

Definition 5.4 (Linear Congruential Generator)**.** The LCG generates a sequence of random numbers via the following linear recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m \tag{5.1}$$

where:

- X_{n+1} is the next number in the sequence.
- X_n is the current number.
- a, c, and m are constants, known as the multiplier, increment, and modulus, respectively.
- $\mod$ denotes the modulus operation.
- The initial or seed value $X_0 = S$, is also required to start the sequence.

The LCG is designed to generate a sequence of numbers that appear random but are deterministically produced by the recurrence relation. This deterministic production makes the sequence reproducible, an essential property in many applications.
The key idea behind the LCG is the modulus operation, which allows the generator to produce a sequence of numbers in a specific range (0 to $m-1$), regardless of the values of a, c, and X_n. The parameters a, c, and m can be carefully chosen to produce sequences with desirable properties. For example, with the right parameters, the LCG can achieve a long period (up to m) before repeating, which is another important characteristic for a good pseudorandom number generator.
The Linear Congruential Generator (LCG) has several key characteristics that shape its suitability as a random number generator.
Beginning with the simplest properties, the LCG is notably reproducible and efficient. Reproducibility is a crucial characteristic in many applications, such as simulations, where repeating the same sequence of numbers is vital for replicating results. With an LCG, one can always expect the same sequence of numbers when provided with the same seed and constants.
When it comes to efficiency, the LCG shines as well. The generation process involves merely multiplication, addition, and modulus operations, all of which are computationally inexpensive. The minimalistic requirement of state space, which is just the last generated number, further enhances this efficiency. This makes LCGs an ideal choice for systems burdened by limited computational resources or memory.
Unpredictability, another essential attribute of a good random number generator, is somewhat of a mixed bag for the LCG. While it's generally challenging to predict the output numbers without knowing the multiplier, increment, modulus, and seed, a person with knowledge of the algorithm and access to a sufficient number of sequential numbers from the sequence can potentially calculate the constants and forecast future numbers. Due to this, LCGs are not recommended for applications where a high level of unpredictability, such as in cryptography, is necessary.
To have an idea of the period, consider LCG with the recurrence relation:

$$X_{n+1} = (3X_n + 5) \mod 8 \tag{5.2}$$

where the seed value $X_0 = 1$. This LCG generates a sequence of integers between 0 and 7. See figure 5.3. For this LCG, the sample space Ω for all possible output is the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$. A period of 8, as in the previous example, is indeed quite small and could introduce noticeable patterns in the generated random numbers. It is clear that the maximum period of the LCG is m.

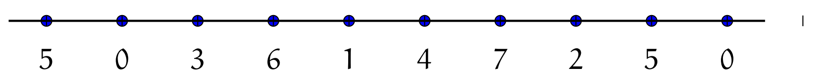


Figura 5.3: A sequence of realizations from a linear congruential generator (LCG). The x-axis represents the index in the sequence, and the y-coordinate of each point represents the value of the LCG at that index. The sequence is shown until it begins to repeat, including the first and second term of the repetition.

Choosing better values for the initial seed and the parameters of the RNG can lead to a longer period. Linear Congruential Generators (LCGs) are a type of pseudorandom number generator characterized by the recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m$$

where:

- X is the sequence of pseudorandom numbers,
- m (the modulus) is $0 < m$,
- a (the multiplier) is $0 < a < m$,
- c (the increment) is $0 \leq c < m$,
- X_0 (the seed) is $0 \leq X_0 < m$.

The choice of m, a, and c significantly influences the quality and period of the pseudorandom number sequence generated by the LCG. Some typically used values for m, a, and c are:

1. **Numerical Recipes:** Uses $m = 2^{32}$, $a = 1664525$, and $c = 1013904223$.
2. **Borland C/C++:** Borland's LCG for its C/C++ compiler uses $m = 2^{32}$, $a = 22695477$, and $c = 1$.
3. **glibc (used in GCC):** The GNU Compiler Collection's C library uses $m = 2^{31}$, $a = 1103515245$, and $c = 12345$.
4. **Java's `java.util.Random`:** Java utilizes $m = 2^{48}$, $a = 25214903917$, and $c = 11$.

These parameters are chosen to maximize the period of the LCG (ideally m, which is the maximum possible period for an LCG) and to ensure a good distribution of the pseudorandom numbers across the available range. The selection of a, c, and m also aims to minimize correlations between successive numbers in the sequence. Proper selection of these constants is crucial for the statistical properties of the generated sequence.

5.3 Generating Discrete Random Variables

Suppose that we want to generate the value of a random variable X having probability mass function

$$p_j = P(X = x_j), \quad j = 0, 1, \ldots$$

This can be accomplished by generating a random number U, and then setting

$$X = \begin{cases} x_0, & \text{if } U < p_0 \\ x_1, & \text{if } p_0 \leq U < p_0 + p_1 \\ \vdots & \\ x_j, & \text{if } \sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i \\ \vdots & \end{cases}$$

Because $P(a \leq U < b) = b - a$, for $0 < a < b < 1$, we have

$$P(X = x_j) = P\left(\sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i\right) = p_j$$

Remark: If the x_i are such that $x_i < x_{i+1}$, then

$$X = x_j \text{ if } F(x_{j-1}) \leq U < F(x_j)$$

where $F(x_k) = \sum_{i=0}^{k} p_i$ is the distribution function of X. Therefore, the value of X is determined by generating a random number U and then determining the interval $(F(x_{j-1}), F(x_j))$ in which U lies. As this is equivalent to finding the inverse of $F(U)$, the preceding method is called the *inverse transform algorithm* for generating X.

Example: Generating a Geometric Random Variable

Recall that X is geometric with parameter p if

$$P(X = j) = pq^{j-1}, \quad j \geq 1$$

where $q = 1 - p$. Such a random variable represents the trial number of the first success when independent trials having a common success probability p are performed in sequence. Because

$$\sum_{i=1}^{j-1} P(X = i) = 1 - P(X > j - 1) = 1 - q^{j-1}$$

we can generate X by generating a random number U and then setting X equal to that value j such that

$$1 - q^{j-1} \leq U < 1 - q^j$$

or, equivalently, such that

$$q^j < 1 - U \leq q^{j-1}$$

That is,

$$X = \text{mín}\{j : q^j < 1 - U\}$$

$$= \text{mín}\{j : j \log(q) < \log(1-U)\}$$
$$= \text{mín}\left\{j : j > \frac{\log(1-U)}{\log(q)}\right\}$$
$$= \left\lfloor \frac{\log(1-U)}{\log(q)} \right\rfloor + 1$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. Because $1-U$ is also uniformly distributed on (0, 1), it follows that

$$X = \left\lfloor \frac{\log(U)}{\log(q)} \right\rfloor + 1$$

is also geometric with parameter p.

Example: Generating a Binomial Random Variable

To generate a binomial random variable with parameters n, p, we make use of the result that its probability mass function

$$p_i = P(X = i) = \binom{n}{i} p^i (1-p)^{n-i}, \quad i = 0, \ldots, n$$

satisfies the identity

$$p_{i+1} = \frac{n-i}{i+1} \cdot \frac{p}{1-p} \cdot p_i$$

Consequently, we can express the inverse transform algorithm as follows:

1. Generate a random number U.
2. Calculate $c = \frac{p}{1-p}$, $i = 0$, $\alpha = (1-p)^n$, $F = \alpha$
3. If $U \leq F$, set $X = i$ and stop.
4. Calculate $\alpha = c\frac{n-i}{i+1}\alpha$, $F = F + \alpha$, $i = i + 1$
5. Go to Step 3.

Example: Generating a Poisson Random Variable

The probability mass function of a Poisson random variable with mean λ

$$p_i = P(X = i) = e^{-\lambda}\frac{\lambda^i}{i!}, \quad i = 0, 1, \ldots$$

is easily shown to satisfy the identity

$$p_{i+1} = \frac{\lambda}{i+1} p_i, \quad i \geq 0 \tag{9.1}$$

Using the preceding recursion equation to compute the Poisson probabilities as they become needed, the inverse transform algorithm for generating a Poisson random variable with mean λ can be expressed as follows. (The quantity i refers to the value under consideration at present, α is the probability that X is equal to i, and F is the probability that X is less than or equal to i.)

1. Generate a random number U.
2. If $i = 0$, $\alpha = e^{-\lambda}$, $F = \alpha$.
3. If $U < F$, set $X = i$ and stop.
4. Otherwise, $\alpha = \frac{\lambda}{i+1}\alpha$, $F = F + \alpha$, $i = i + 1$.
5. Go to Step 3.

To check that the preceding algorithm does indeed generate a Poisson random variable with mean λ, note that it first generates a random number U and then checks whether $U < e^{-\lambda} = p_0$. If so, it sets $X = 0$; if not, it computes p_1 by using the recursion Equation (9.1), and then checks whether $U < p_0 + p_1$, and so on.

5.4 Generating continuous random variables

By using the inverse transform sampling method, we can simulate any continuous random variable, provided we can find the inverse of its CDF. This method is widely used in simulation studies, statistical modeling, and machine learning applications.

Theorem 5.5. If $U \sim U(0, 1)$ and $F(\cdot)$ is a valid invertible cumulative distribution function (CDF), then

$$X = F^{-1}(U)$$

Demostración.

$$\begin{aligned} P(X \le x) &= P(F^{-1}(U) \le x) \\ &= P(U \le F(x)) \\ &= F_U(F(x)) \\ &= F(x). \end{aligned}$$

□

This theorem provides a straightforward way to generate numbers from an arbitrary probability distribution by simulating uniform distriuted numbers and calculating the proper transformation.

Example. The exponential distribution with rate parameter $\lambda > 0$ has the probability density function (pdf):

$$p(x; \lambda) = \lambda \exp(-\lambda x) \quad \text{for } x \ge 0. \tag{5.3}$$

The cumulative distribution function (CDF) of the exponential distribution is:

$$F(x; \lambda) = 1 - \exp(-\lambda x) \quad \text{for } x \ge 0. \tag{5.4}$$

The inverse of the CDF is:

$$F^{-1}(y;\lambda) = -\frac{1}{\lambda}\log(1-y) \quad \text{for } 0 < y < 1. \tag{5.5}$$

To generate a random sample from the exponential distribution using the inverse transform method, we first generate a random sample y from the uniform distribution on the interval (0, 1), and then apply the inverse CDF to y:

$$x = F^{-1}(y;\lambda) = -\frac{1}{\lambda}\log(1-y). \tag{5.6}$$

This x is a random sample from the exponential distribution with rate parameter λ.

Theorem 5.6. Realizations of X can be generated by simulation $U \sim U(0,1)$, and then

$$X = min\{x|F_X(x) \geq U\}$$

Demostración.

$$\begin{aligned} P(X = x) &= P_{(U)}(U \in (P_X(X \leq x-1), P_X(X \leq x)]) \\ &= P_X(X \leq x-1) - P_X(X \leq x) \\ &= P_X(X = x). \end{aligned}$$

□

Simulation of NHPP Using the Thinning Method

The thinning method is a simulation technique for generating a path of an NHPP with a given rate function $\lambda(t)$. The method is implemented through the following steps:

1. Establish a constant λ^* such that $\lambda^* \geq \sup_{t\geq 0}\lambda(t)$.
2. Generate a sequence of independent exponential random variables $X_1, X_2, \ldots$ with rate λ^*, representing the inter-event times in a homogeneous Poisson process.
3. Simultaneously, generate a sequence of independent uniform random variables $U_1, U_2, \ldots$, each uniformly distributed over [0,1].
4. Accumulate the times $T_i = \sum_{j=1}^{i} X_j$ to find the time of the ith event. If T_i exceeds the period of interest, terminate the simulation.
5. Retain each event T_i with probability $\frac{\lambda(T_i)}{\lambda^*}$, effectively 'thinning' the homogeneous Poisson process to match the desired nonhomogeneous intensity function $\lambda(t)$.

Exercises

1: The negative binomial distribution with its probability mass function is defined as:

$$P(X=k)=\binom{k-1}{r-1}p^r(1-p)^{k-r}, \quad k=r,r+1,r+2,\dots$$

where k is the total number of trials required to achieve r successes, p is the probability of success on each trial, and the trials are independent.

Give a method for simulating a negative binomial random variable.

2: Give an algorithm for simulating a random variable having density function

$$f(x)=30(x^2-2x^3+x^4), \quad 0<x<1$$

3: Suppose we want to simulate X having probability mass function $P\{X=i\}=P_i, i=1,\dots,n$ and suppose we can easily simulate from the probability mass function Q_i, $\sum_{i=1}^{n} Q_i = 1, Q_i \geq 0$. Let C be such that $P_i \leq CQ_i, i=1,\dots,n$. Show that the following algorithm, named **the discrete rejection method**, generates the desired random variable:

Step 1: Generate Y having mass function Q and U an independent random number.

Step 2: If $U \leq \frac{P_Y}{CQ_Y}$, set $X=Y$. Otherwise return to step 1.

4: Give another algorithm for generating a binomial random variable with parameters n, p, by recalling how such a random variable arises. Compare it with the inverse transform algorithm. Which one do you think is quicker?

Stochastic Processes

6.1 Continuous-Time Markov Chains

A continuous-time Markov chain (CTMC) is a type of stochastic process characterized by jumps from one state to another with sojourn times that are exponentially distributed. This property aligns with the process's Markovian nature, which implies no memory of past states beyond the current state. The foundational property of a CTMC can be mathematically expressed as:

$$P(T_i > s + t \mid T_i > s) = P(T_i > t), \quad \text{for all } s, t \geq 0. \tag{6.1}$$

The survival function $S(t)$ is defined by the probability $S(t) = P(T > t)$, where T is the random variable representing the time until transition in a continuous-time Markov chain. The memoryless property of this process is expressed as:

$$P(T > s + t \mid T > s) = P(T > t)$$

Using the definition of conditional probability, this can be rewritten as:

$$\frac{P(T > s + t)}{P(T > s)} = P(T > t)$$

Substituting the survival function $S(t) = P(T > t)$ into this equation leads to:

$$\frac{S(s + t)}{S(s)} = S(t)$$

Rearranging gives the functional equation for the survival function:

$$S(s + t) = S(s)S(t)$$

Memoryless Property and Exponential Distribution

The memoryless property states that the probability of the process remaining in its current state does not depend on how long it has already been in that state. This characteristic can only be satisfied by the exponential distribution. We can demonstrate this by considering the following reasoning:

1. **Memorylessness Definition**: For a random variable T, representing the time until the chain transitions from state i, memorylessness implies:

$$P(T > s + t \mid T > s) = P(T > t). \tag{6.2}$$

2. **Characterizing the Exponential Distribution**:

    - Assume T has a cumulative distribution function $F(t) = P(T \leq t)$.

- The survival function, $S(t) = P(T > t) = 1 - F(t)$, describes the probability that the process has not transitioned out of state i by time t.
- From memorylessness, $S(s + t) = S(s)S(t)$.
- The only continuous distribution satisfying this functional equation is the exponential distribution, where $S(t) = e^{-\lambda t}$ for some $\lambda > 0$.

3. **Rate of Transition**:

   - The rate parameter λ of the exponential distribution, often denoted as ν_i for state i, is the reciprocal of the expected sojourn time in state i, $\frac{1}{\nu_i}$.
   - Transitions between states are governed by probabilities p_{ij}, where $j \neq i$.

Independence of Sojourn Time and Next State

An essential aspect of CTMCs is the independence between the sojourn time in state i and the choice of the next state j. This independence is vital because any dependency would violate the Markov property by introducing memory effects that depend on the duration in the current state, which is contrary to the defining characteristics of Markov processes.
[Computer Algorithm Process Model] Consider a computational routine with two sequential processes: Process 1 and Process 2. The routine begins with Process 1, which performs the initial computation, followed by Process 2, which completes the computation with further refinement. Let the execution times for each process be independent exponentially distributed random variables with rates μ_1 for Process 1 and μ_2 for Process 2, respectively. Let's also assume that tasks arrive following a Poisson process with a rate of λ, and a new task is only initiated if both processes are idle.
To model this as a continuous-time Markov chain, we define the state space to reflect the computational pipeline:

- State 0: Both processes are idle, waiting for tasks.
- State 1: Process 1 is active, working on a task.
- State 2: Process 1 is finished, and Process 2 is active.

The system toggles between being idle and processing a task, with no new tasks starting until the current one is completely processed by both stages. The transition rates are denoted by

- $\nu_0 = \lambda$: Rate at which new tasks start when both processes are idle.
- $\nu_1 = \mu_1$, $\nu_2 = \mu_2$: Rates at which Process 1 and Process 2, respectively, complete tasks.
- $P_{01} = 1$: Once a task starts, it certainly goes to Process 1.
- $P_{12} = 1$: After completion of Process 1, the task definitely moves to Process 2.
- $P_{20} = 1$: Following Process 2, the task always leaves the system, returning it to an idle state.

6.2 Birth-Death Process

A Birth-Death process is a specific type of continuous-time Markov chain often used to model population dynamics, queuing systems, and chemical reactions.

Definition 6.1 (Birth-Death Process). A Birth-Death process is a continuous-time Markov chain $\{X(t) : t \geq 0\}$ characterized by birth rates λ_n and death rates μ_n depending on the current state n. The process evolves as follows:

- From state n, it transitions to state $n+1$ with rate λ_n.
- From state n, it transitions to state $n-1$ with rate μ_n.

Birth-Death processes are widely used in ecology to model population growth and decline, in computer science to model queuing systems, and in epidemiology to model the spread of diseases.

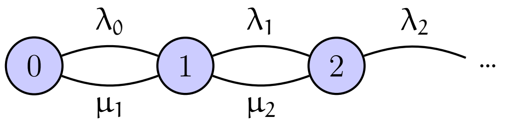


Figura 6.1: Diagram illustrating a Birth-Death Process

Consider a system represented by the number of individuals in it at any given time. The system follows a birth and death process, where new arrivals occur at rate λ_n and departures at rate μ_n when there are n individuals present. The times between arrivals and departures are exponentially distributed with means $1/\lambda_n$ and $1/\mu_n$, respectively. This model is a continuous-time Markov chain with state space $\{0, 1, 2, \dots\}$ and transitions between state n and $n \pm 1$. The transition rates and probabilities are given by:

$$\begin{aligned}
\nu_0 &= \lambda_0, & \\
\nu_i &= \lambda_i + \mu_i, & i > 0, \\
P_{01} &= 1, & \\
P_{i,i+1} &= \frac{\lambda_i}{\lambda_i + \mu_i}, & i > 0, \\
P_{i,i-1} &= \frac{\mu_i}{\lambda_i + \mu_i}, & i > 0.
\end{aligned}$$

The model assumes that the system transitions from state n to $n+1$ if a birth occurs before a death, and the probability of an exponential random variable with rate λ_i occurring before one with rate μ_i is $\lambda_i/(\lambda_i + \mu_i)$.

Example (The Poisson Process). Consider a birth and death process where there are no deaths, i.e., $\mu_n = 0$ for all $n \geq 0$, and the birth rate $\lambda_n = \lambda$ for all $n \geq 0$. This process is then simply the Poisson process with the time between arrivals being exponentially distributed with mean $1/\lambda$.

Example (Birth Process with Linear Birth Rate)**.** In a population where each individual independently gives birth at an exponential rate λ, the total birth rate is $\lambda_n = n\lambda$ when there are n individuals. This is known as a Yule process.

Example (Linear Growth Model with Immigration)**.** For a population where each individual gives birth at a rate λ, and there is an immigration rate θ, the total birth rate is $\lambda_n = n\lambda + \theta$. Death occurs at a rate $\mu_n = n\mu$.

Example (Machine Repair Model)**.** Consider M machines with a single repair person. Machines break down at an exponential rate $1/\lambda$, and repairs are completed at a rate $1/\mu$. The process is in state n when n machines are not working, and transitions occur as machines break down or are repaired.

Example (Two-State Continuous-Time Markov Chain)**.** A machine works for an exponential time with mean $1/\lambda$ before failure and is repaired in an exponential time with mean $1/\mu$. With state 0 representing a working machine and state 1 representing a machine being repaired, this system is modeled by a two-state continuous-time Markov chain.

6.2.1. Gillespie Algorithm for Species Evolution

The Gillespie Algorithm, traditionally used in chemical kinetics, is adeptly adapted here for modeling stochastic speciation and extinction events in species evolution. This approach is particularly useful in ecological studies where the population dynamics are influenced significantly by random discrete events.

Procedure of the Gillespie Algorithm

The Gillespie Algorithm simulates the sequence and timing of speciation and extinction events based on current population state, with the following steps:

1. **Initialization**: Set the initial number of species $N(0)$ and start the simulation at time $t = 0$.

2. **Rate Calculation**: Calculate the total rates of speciation (λ) and extinction (μ) based on the current number of species $N(t)$. Each species has a chance to speciate or go extinct, thus the total rates are:

$$\lambda_{\text{total}} = N(t) \times \lambda, \quad \mu_{\text{total}} = N(t) \times \mu$$

   The total rate of any event occurring is $a_0 = \lambda_{\text{total}} + \mu_{\text{total}}$.

3. **Determine Waiting Time**: Compute the time Δt until the next event occurs using an exponential distribution:

$$\Delta t = \frac{1}{a_0} \ln\left(\frac{1}{r}\right),$$

where r is a random number uniformly distributed between 0 and 1.

4. **Event Selection**: Determine the next event type (speciation or extinction) by sampling from a multinomial distribution. The probabilities are proportional to the rates:

$$P(\text{Speciation}) = \frac{\lambda_{\text{total}}}{a_0}, \quad P(\text{Extinction}) = \frac{\mu_{\text{total}}}{a_0}.$$

5. **Update System**: Adjust the number of species $N(t)$ according to the type of event. Increase by one for speciation, decrease by one for extinction.

6. **Advance Time**: Update the simulation time by Δt, setting $t = t + \Delta t$.

7. **Repeat**: Continue the simulation until a stopping criterion is met, such as reaching a specified final time or a critical population threshold.

Example

Example. Imagine a small island ecosystem starting with 10 species. The speciation rate λ is 0.1 per year per species, and the extinction rate μ is 0.05 per year per species. Using the Gillespie Algorithm, one can simulate the number of species over 100 years to study possible extinction scenarios or flourishing of biodiversity.

6.3 Renewal Theory

Renewal Theory is a branch of probability theory that deals with the times at which events recur. It is particularly useful in processes where events occur repeatedly over time at random intervals, such as in queueing systems, reliability engineering, and stochastic modeling for biological or financial applications.

A Renewal Process is a sequence of times at which a given event (such as a system failure and repair, or customer arrival) occurs. Mathematically, let $\{X_n\}_{n=1}^{\infty}$ be a sequence of independent and identically distributed (i.i.d.) positive random variables representing the times between consecutive events (inter-arrival times). The times of occurrences of the events are then given by the partial sums:

$$S_0 = 0, \quad S_n = \sum_{i=1}^{n} X_i \quad \text{for } n \geq 1,$$

where S_n is the time of the n-th event.

Basic Concepts in Renewal Theory

- **Renewal Function:** The Renewal Function $M(t)$ represents the expected number of times the event has occurred by time t, defined as:

$$M(t) = \mathbb{E}[\text{Number of renewals by time } t] = \sum_{n=0}^{\infty} P(S_n \leq t).$$

- **Inter-arrival Distribution:** The times between events are described by their probability distribution, which significantly influences the behavior of the Renewal Process. Common distributions include exponential, gamma, and Weibull distributions, each suitable for different types of applications.

- **Renewal Equation:** The Renewal Equation is a key relation in Renewal Theory, relating the renewal function with the inter-arrival distribution. It is given by:

$$m(t) = F(t) + \int_0^t m(t-x)\,dF(x),$$

where $F(t)$ is the cumulative distribution function of the inter-arrival times and $m(t)$ is the density function of the renewal function $M(t)$.

Applications of Renewal Theory

Renewal Theory provides the mathematical foundation for analyzing and predicting the behavior of systems where events happen recurrently over time. Applications include:

- **Queueing Theory:** Modeling the arrival of customers in a service system.

- **Reliability Engineering:** Assessing product lifetimes and maintenance schedules.

- **Inventory Management:** Determining restocking times based on usage rates.

- **Biological Systems:** Modeling recurrent biological phenomena such as heartbeats or neural activity.

Limit Theorems in Renewal Theory

Two fundamental results in Renewal Theory are the Elementary Renewal Theorem and the Key Renewal Theorem, which provide long-term insights about the behavior of renewal processes:

- **Elementary Renewal Theorem:** States that the average number of renewals in a unit time interval stabilizes as time goes to infinity:

$$\operatorname{lím}_{t\to\infty} \frac{M(t)}{t} = \frac{1}{\mu},$$

where μ is the expected value of the inter-arrival time X_1.

- **Key Renewal Theorem:** Provides a more general conclusion, which is useful for non-identical distributions and complex scenarios. It is particularly valuable in deriving steady-state behaviors in stochastic processes:

$$\operatorname{lím}_{t\to\infty} \sum_{n=0}^{\infty} f(t-S_n) = \left(\int_0^\infty f(u)\,du\right)\left(\frac{1}{\mu}\right),$$

for directly Riemann integrable function f, showing that the process stabilizes according to the mean inter-arrival time.

6.4 Queuing Theory

Queuing Theory is the mathematical study of waiting lines, or queues. It uses models to show the process of arriving at the queue, waiting in the queue (if any), and being served by the server(s). This theory is applicable in various fields such as telecommunications, traffic engineering, computing, and in the design and operation of factories, shops, offices, and hospitals.
These processes are birth-death processes, characterized by the rates at which new customers arrive (births) and existing customers are served (deaths).

Definition

Let $\{X(t)\}_{t\geq 0}$ be a continuous-time Markov chain with state space $\{0, 1, 2, \ldots\}$. The process $X(t)$ is a birth-death process if transitions can only occur between neighboring states, specifically:

$$P(X(t+\Delta t) = n+1 \mid X(t) = n) = \lambda_n \Delta t + o(\Delta t),$$

$$P(X(t+\Delta t) = n-1 \mid X(t) = n) = \mu_n \Delta t + o(\Delta t),$$

where λ_n is the birth rate and μ_n is the death rate.

Steady-State Solution

The steady-state probabilities π_n are the long-run probabilities that the system is in state n. These probabilities satisfy the balance equations:

$$\lambda_{n-1}\pi_{n-1} = \mu_n \pi_n, \quad n \geq 1,$$

and the normalization condition:

$$\sum_{n=0}^{\infty} \pi_n = 1.$$

Solving the balance equations, we get:

$$\pi_n = \pi_0 \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}}, \quad n \geq 1,$$

where π_0 is determined by the normalization condition:

$$\pi_0 \left(1 + \sum_{n=1}^{\infty} \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}} \right) = 1.$$

Queuing Networks

A queuing network consists of multiple interconnected queues. Customers move between queues, and the performance of the entire network is of interest.

Open Queuing Networks: Customers can enter from outside the network, pass through one or more service nodes, and then leave the network. The arrival process is usually modeled as a Poisson process.

Closed Queuing Networks: The total number of customers is fixed. Customers circulate within the network, moving from one service node to another.

Jackson Networks

Jackson Networks are a class of queuing networks where the nodes are interconnected such that the arrival process to each node is a Poisson process, and the service times are exponentially distributed.

Steady-State Distribution: For a network with K nodes, let λ_i be the effective arrival rate at node i and μ_i be the service rate at node i. The steady-state probability $P(\mathbf{n})$ of having n_i customers at node i (for $\mathbf{n} = (n_1, n_2, \ldots, n_K)$) is given by:

$$P(\mathbf{n}) = \prod_{i=1}^{K} \left(\frac{\rho_i^{n_i}}{(1-\rho_i)} \right),$$

where $\rho_i = \frac{\lambda_i}{\mu_i}$ is the utilization of node i.

Performance Measures

Typical performance measures in queuing networks include:

Average number of customers L in the system or queue:

$$L = \sum_{n=0}^{\infty} n\pi_n.$$

Average time W a customer spends in the system:

$$W = \frac{L}{\lambda_{\text{eff}}},$$

where λ_{eff} is the effective arrival rate.

Utilization ρ of each server:

$$\rho = \frac{\lambda}{\mu}.$$

6.5 Reliability Theory

Reliability Theory is a field of study that deals with the ability of a system or component to function under stated conditions for a specified period. It is widely used in engineering, quality control, and risk assessment to predict and improve the performance and safety of products and systems.

System Reliability

System reliability refers to the probability that a system will perform its intended function for a given period under specified conditions. Systems can be composed of multiple components arranged in series, parallel, or combinations of both.

Series Systems

In a series system, all components must function for the system to function. If any component fails, the entire system fails. Let $R_i(t)$ be the reliability of component i at time t. The reliability of a series system with n components is:

$$R_{\text{series}}(t) = \prod_{i=1}^{n} R_i(t).$$

Parallel Systems

In a parallel system, the system functions as long as at least one component functions. Let $R_i(t)$ be the reliability of component i at time t. The reliability of a parallel system with n components is:

$$R_{\text{parallel}}(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)).$$

Complex Systems

For systems with combinations of series and parallel components, reliability can be determined using methods such as reliability block diagrams or fault tree analysis. The overall system reliability is a function of the reliabilities of individual components and their configuration.

Failure Models

Failure models describe the statistical behavior of component lifetimes. These models are essential for predicting and improving system reliability.

Applications of Reliability Theory

Reliability Theory is used in various applications to ensure that systems are dependable and to optimize maintenance schedules.

- **Reliability Engineering:** Assessing and improving product lifetimes and maintenance schedules.
- **Quality Control:** Monitoring and improving manufacturing processes to reduce defects and failures.
- **Risk Assessment:** Evaluating the reliability and safety of critical systems such as aerospace, nuclear power, and medical devices.

6.6 Brownian Motion

Brownian Motion is a continuous-time stochastic process that models random movement, often used to describe the erratic behavior of particles suspended in a fluid. It has significant applications in physics, finance, and various fields of engineering.

Wiener Process

The Wiener process, named after Norbert Wiener, is the mathematical formalization of Brownian Motion. It is a fundamental process in stochastic calculus and has the following properties:

Definition

A Wiener process $\{W(t)\}_{t\geq 0}$ is a continuous-time stochastic process that satisfies the following properties:

1. $W(0) = 0$.
2. $W(t)$ has independent increments.
3. $W(t) - W(s) \sim N(0, t-s)$ for $0 \leq s < t$.
4. $W(t)$ has continuous paths.

Properties

Markov Property The future evolution of the process depends only on the current state and not on the past history:

$$P(W(t+h) \leq x \mid \mathcal{F}_t) = P(W(t+h) \leq x \mid W(t)),$$

where $\mathcal{F}_t$ is the filtration representing the history up to time t.

Martingale Property The Wiener process is a martingale, meaning:

$$E[W(t) \mid \mathcal{F}_s] = W(s), \quad \text{for } 0 \leq s \leq t.$$

Quadratic Variation The quadratic variation of $W(t)$ over the interval $[0, t]$ is:

$$[W](t) = t.$$

Applications in Finance

Brownian Motion is extensively used in financial mathematics to model the behavior of asset prices and to derive various financial instruments.

Geometric Brownian Motion (GBM)

A widely used model for stock prices is the Geometric Brownian Motion (GBM), which assumes that the logarithm of the stock price follows a Brownian Motion with drift. The stochastic differential equation (SDE) for GBM is:

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t,$$

where S_t is the stock price at time t, μ is the drift rate, σ is the volatility, and W_t is a Wiener process.

Solution to GBM The solution to this SDE is given by:

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right),$$

where S_0 is the initial stock price.

Black-Scholes Model

The Black-Scholes model for option pricing is derived using GBM. The price of a European call option is given by the Black-Scholes formula:

$$C(S_t, t) = S_t \Phi(d_1) - K e^{-r(T-t)} \Phi(d_2),$$

where:

$$d_1 = \frac{\ln(S_t/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t},$$

Φ is the cumulative distribution function of the standard normal distribution, K is the strike price, T is the maturity, and r is the risk-free interest rate.

Applications in Physics

In physics, Brownian Motion describes the random motion of particles suspended in a fluid, providing insights into the kinetic theory of gases and the behavior of microscopic particles.

Einstein's Explanation

Albert Einstein's explanation of Brownian Motion provided a theoretical foundation for the kinetic theory of gases. He related the mean squared displacement of a particle to time and temperature:

$$\langle x^2 \rangle = 2Dt,$$

where D is the diffusion coefficient.

Langevin Equation

The Langevin equation provides a description of the dynamics of particles under the influence of random forces:

$$m\frac{d^2x}{dt^2} = -\gamma\frac{dx}{dt} + \eta(t),$$

where m is the mass of the particle, γ is the friction coefficient, and $\eta(t)$ is a random force representing the thermal fluctuations.

Applications in Engineering

Brownian Motion is used in engineering to model noise and fluctuations in systems, providing a framework for designing systems that can operate reliably under random perturbations.

Stochastic Control

In control engineering, stochastic control involves designing controllers that can handle systems with inherent randomness. The Linear Quadratic Gaussian (LQG) controller is an example that uses a combination of linear dynamics, Gaussian noise, and quadratic cost functions.

Signal Processing

In signal processing, Brownian Motion models can be used to describe the behavior of noise in electronic circuits and communication systems. Techniques such as Kalman filtering are used to estimate the state of a system in the presence of random noise.

Mathematical Tools

Advanced applications of Brownian Motion often require sophisticated mathematical tools, including:

Ito Calculus

Ito calculus extends the standard calculus to handle stochastic integrals. The Ito Lemma is a fundamental result used to find the differential of a function of a stochastic process:

$$df(X_t) = f'(X_t)\,dX_t + \frac{1}{2}f''(X_t)\,d[X]_t,$$

where $[X]_t$ is the quadratic variation of X_t.

Stochastic Differential Equations (SDEs)

SDEs are used to model systems influenced by random noise. They take the form:

$$dX_t = \mu(X_t, t)\,dt + \sigma(X_t, t)\,dW_t,$$

where μ is the drift term and σ is the diffusion term.

Stochastic Modeling

Definition 7.1 (Program). A **program** P is defined as a tuple (M, θ), where:

- $M : I \to O$ is a function from the input space I to the output space O. M encapsulates the operational logic of the program, defining specific computations to transform each input $i \in I$ into an output $o \in O$.
- $\theta \in \Theta$ are the parameters of the model M, with Θ representing the parameter space. These parameters adjust the function M, allowing it to be specialized or generalized according to different tasks or datasets.

In operational terms, for a given input $x \in I$, the program applies M parameterized by θ to produce an output y, expressed as $y = M_\theta(x)$. This computation models the essential mechanism by which the program processes data and generates results.

Example (Sorting Program). Let $I = \{x \mid x \text{ is a list of integers}\}$ and $O = \{y \mid y \text{ is a list of integers sorted}$
Define a sorting program with $M : I \to O$ such that for any $x \in I$, $M(x) = y$, where y is the permutation of x sorted in non-decreasing order.

Another example involves image recognition tasks. Let's consider an image as a matrix:

Example (Dog/Cat Identification Program). Consider Im as a matrix representing a grayscale image, where each element Im_{ij} denotes the pixel intensity at position (i, j). Let $I = \{\text{Im} \mid \text{Im is a } m \times n \text{ matrix}\}$ and $O = \{'\mathtt{dog}', '\mathtt{cat}'\}$. Define a classification program with $M : I \to O$ trained to map an input image $\text{Im} \in I$ to a label $y \in O$ based on its content.

These examples demonstrate how a program as defined by the function M and its parameters θ, effectively maps inputs to outputs within specified computational domains.

7.1 Linear Regression

Linear regression is a fundamental statistical method for prediction, modeling the relationship between a dependent variable and one or more explanatory variables.
Consider a dataset $\{(u^{(i)}, v^{(i)})\}_{i=1}^N$, with each $u^{(i)} \in \mathbb{R}^p$ as a vector of p input features, and $v^{(i)} \in \mathbb{R}$ as the corresponding observed output. We define the model function M as:

$$M_\theta(u^{(i)}) = \theta_0 + \sum_{j=1}^{p} \theta_j u_j^{(i)},$$

where $\theta = (\theta_0, \theta_1, \ldots, \theta_p)$ are the model parameters, including the intercept θ_0 and coefficients θ_j for each feature.
The goal of linear regression is to find the parameter set θ that minimizes the sum of squared residuals, effectively fitting the model to the data. This objective is formulated as:

$$\min_{\theta} \sum_{i=1}^{N} \left(v^{(i)} - M_\theta(u^{(i)})\right)^2.$$

To address this optimization problem using the least squares method, the design matrix U, which facilitates the solution, is structured as follows:

$$U = \begin{bmatrix} 1 & u_1^{(1)} & u_2^{(1)} & \cdots & u_p^{(1)} \\ 1 & u_1^{(2)} & u_2^{(2)} & \cdots & u_p^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & u_1^{(N)} & u_2^{(N)} & \cdots & u_p^{(N)} \end{bmatrix},$$

where each row corresponds to an input vector $u^{(i)}$, augmented with a leading 1 to account for the intercept θ_0.
The least squares solution to the optimization problem is then computed as:

$$\theta^* = (U^T U)^{-1} U^T v, \tag{7.1}$$

where v is the vector of observed outputs.
So, given data, we can 'train' a model directly using equation 7.1 and obtain a program, to predict the outcome of future input features.

7.2 Logistic Regression

Logistic regression is a key method for binary classification, predicting probabilities of binary outcomes effectively. This approach is particularly suitable for scenarios where the response variable is categorical, such as in medical diagnostics, spam detection, or credit scoring.
Consider a dataset $\{(u^{(i)}, v^{(i)})\}_{i=1}^{N}$, where each $u^{(i)} \in \mathbb{R}^p$ is a vector of p input features and $v^{(i)} \in \{0, 1\}$ denotes the binary outcomes. The logistic regression model function M is defined as:

$$M_\theta(u^{(i)}) = \sigma(\theta^T u^{(i)}),$$

where

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

is the logistic sigmoid function that converts a linear combination of inputs into a probability.
Probability Model and Loss Function: The outcome $v^{(i)}$ is modeled as a Bernoulli random variable with the success probability $p^{(i)} = M_\theta(u^{(i)})$. The probability of observing $v^{(i)}$ given $u^{(i)}$ under model parameters θ is given by:

$$p(v^{(i)} \mid u^{(i)}, \theta) = (p^{(i)})^{v^{(i)}} (1 - p^{(i)})^{1 - v^{(i)}}.$$

The likelihood function $L(\theta)$, representing the joint probability of observing all the $v^{(i)}$ given $u^{(i)}$ for $i = 1, \ldots, N$, is:

$$L(\theta) = \prod_{i=1}^{N} (p^{(i)})^{v^{(i)}} (1 - p^{(i)})^{1 - v^{(i)}},$$

and the log likelihood, which we seek to maximize, is:

$$\ell(\theta) = \sum_{i=1}^{N} \left[\nu^{(i)} \log(p^{(i)}) + (1 - \nu^{(i)}) \log(1 - p^{(i)}) \right].$$

The goal is to find θ that maximizes $\ell(\theta)$, or equivalently minimizes the negative of $\ell(\theta)$:

$$\min_{\theta} -\ell(\theta) = -\sum_{i=1}^{N} \left[\nu^{(i)} \log(p^{(i)}) + (1 - \nu^{(i)}) \log(1 - p^{(i)}) \right].$$

Optimization and Model Training: This optimization problem is generally solved using iterative techniques such as gradient descent, where each step aims to adjust θ to reduce the negative log likelihood, improving the model fit to the data.

Upon training, the logistic regression model M can then be used as a predictive tool, estimating the probability that new inputs fall into one of the binary categories. This capability makes logistic regression a powerful class of programs within the framework of stochastic modeling for prediction.

Example (Predicting Programming Project Success). Logistic regression is employed to predict the success of programming projects based on evaluations from two experts. Each expert assesses the projects independently, and the logistic model predicts the probability of project success using these assessments.
Given several programming projects evaluated by the experts, the logistic regression model is formulated as:

$$p(\text{Success}) = \frac{1}{1 + e^{-z}},$$

where z is a linear combination of the expert ratings:

$$z = \beta_0 + \beta_1 \text{Expert}_1 + \beta_2 \text{Expert}_2.$$

Model Training: The model is trained using a dataset where each project is labeled as '1' (success) or '0' (failure) based on historical outcomes. The coefficients β_1 and β_2 are optimized to minimize the prediction error, using methods like gradient descent.
Prediction: The success probability $p(\text{Success})$ for each project is calculated. Projects with $p(\text{Success}) \geq 0{,}5$ are predicted as likely to succeed.
Decision-making:

- Projects predicted to succeed may receive additional resources or prioritization.
- Projects with low success probabilities might be reviewed or modified.

7.3 Support Vector Machines

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. The primary objective in SVM is to establish a hyperplane that maximally separates the classes in the feature space.

Cuadro 7.1: Expert Ratings and Project Outcomes

Project ID	Expert 1	Expert 2	Outcome (Success=1)
Project A	8	5	1
Project B	6	5	0
...	...	...	...
Project N	3	9	?

Definition and Composition of SVM

Definition 7.2 (Support Vector Machine)**.** A **Support Vector Machine** M is characterized by a separating hyperplane which divides the feature space I into classes that maximize the margin between the nearest members of different classes, which are termed as support vectors.

Mathematical Model

The optimal hyperplane can be represented by the equation:

$$\mathbf{w}^T\mathbf{x} + b = 0,$$

where $\mathbf{w}$ is the weight vector perpendicular to the hyperplane, $\mathbf{x}$ is the input feature vector, and b is the bias.
The objective is to maximize the margin between the hyperplane and the nearest training data point of any class, which is mathematically given by:

$$\max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|},$$

subject to the constraints for each data point $(\mathbf{x}_i, y_i)$ in the training set:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i.$$

Kernel Trick

For non-linear classification, SVMs use the kernel trick to transform the input space into a higher-dimensional space where a linear separator might exist. The kernel function K relates to the dot product of two feature vectors in this new space:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j),$$

where ϕ is the transformation function to the higher-dimensional space.

Optimization and Learning

The optimization problem in SVMs involves minimizing a quadratic function subject to linear constraints, which is typically solved using methods such as Sequential Minimal Optimization (SMO). Once the model parameters $\mathbf{w}$ and b are determined, the SVM model M acts as a program that assigns new inputs $\mathbf{x}$ to one of the categories based on the sign of $\mathbf{w}^T\mathbf{x} + b$.

Implementation in Data Analysis

In practice, SVMs are used to solve a variety of real-world problems such as image classification, voice recognition, and biological classification, demonstrating robust performance across a wide range of data types and complexities.
This structured approach allows SVMs to model complex relationships in data by maximizing the margin between class boundaries, providing a strong predictive performance characterized by generalization to unseen data.

Generalized Additive Models

Definition 7.3 (Generalized Linear Models)**.** Generalized Linear Models (GLMs) are a broad class of models that extend traditional linear regression by allowing for a response variable $v^{(i)}$ whose distribution is a member of the exponential family, and by utilizing a link function to relate the expected value of the response to the linear predictor. This flexibility allows GLMs to handle a wider range of data types than standard linear regression.

Consider a dataset $\{(u^{(i)}, v^{(i)})\}_{i=1}^N$, with each $u^{(i)} \in \mathbb{R}^p$ representing a vector of p input features, and $v^{(i)}$ representing the observed responses. The systematic component of a GLM, or the linear predictor, is defined as:

$$\eta^{(i)} = \theta^T u^{(i)},$$

where $\eta^{(i)}$ functions as a combination of the input features weighted by the model parameters θ. The link between the mean of the response distribution $\mu^{(i)} = E(v^{(i)})$ and the linear predictor is established through a link function g:

$$\mu^{(i)} = g^{-1}(\eta^{(i)}).$$

Link Function and Response Distribution: The choice of the link function, such as the logit for the binomial distribution or the natural log for the Poisson distribution, is crucial and depends on the nature of the response variable. This ensures the appropriateness of the model to the data and allows for the modeling of variables that are counts, proportions, or times until an event.
Loss Function and Estimation: The estimation of parameters in GLMs is generally performed through maximum likelihood estimation (MLE). The likelihood function, based on the chosen exponential family distribution, is:

$$L(\theta) = \prod_{i=1}^{N} f(v^{(i)} \mid u^{(i)}; \theta),$$

where f is the probability density or mass function. The corresponding loss function, utilized during the optimization process, is the negative log-likelihood:

$$\min_{\theta} -\log L(\theta) = -\sum_{i=1}^{N} \log f(v^{(i)} \mid u^{(i)}; \theta).$$

Generalized Additive Models (GAMs): As a subclass of GLMs, Generalized Additive Models (GAMs) further generalize the linear predictor to include non-linear terms via smooth functions. In GAMs, the linear predictor may include terms like:

$$\eta^{(i)} = \beta_0 + s_1(u_1^{(i)}) + s_2(u_2^{(i)}) + \cdots + s_p(u_p^{(i)}),$$

where s_j are smooth functions of the predictors that capture non-linear relationships. This addition enhances the model's ability to capture complex patterns in the data, making GAMs particularly useful in ecological and environmental modeling.
Optimization and Practical Implementation: Optimization methods such as Newton-Raphson or iterative re-weighted least squares (IRLS) are commonly employed to find the best-fitting model parameters θ that minimize the loss function. This process is key to ensuring that the GLM or GAM accurately reflects the underlying data relationships.

Once trained, both GLM and GAM function as predictive programs, mapping input features to predicted outcomes effectively. This capability makes them invaluable tools for diverse applications in predictive modeling across numerous scientific disciplines.

7.4 Neural Networks

Neural networks are sophisticated mathematical frameworks designed to emulate certain processing patterns found in biological neural systems. Below, we describe the foundational components and the mathematical structure of neural networks.

General Structure

Neural networks consist of layers of interconnected nodes called neurons. Each neuron in a layer is connected to several other neurons in the previous and next layers, forming a network that can propagate data forward from input to output.

Definition 7.4 (Neural Network Architecture). A Neural Network M is formally defined by its architecture, which includes layers of neurons, and a set of weights $\mathbf{w}$. The architecture determines how inputs are transformed through the network:

$$M(\mathbf{x}; \mathbf{w}) = f_L \circ \cdots \circ f_1(\mathbf{x}; \mathbf{w}_1, \ldots, \mathbf{w}_L)$$

where $\mathbf{x}$ is the input vector to the network, $\mathbf{w}_i$ denotes the weights for the i-th layer, and f_i are the transformation functions specific to each layer.

Key Operations

Linear Combination: Each neuron computes a weighted sum of its inputs, which serves as the input to a non-linear activation function. The output of each neuron is given by:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

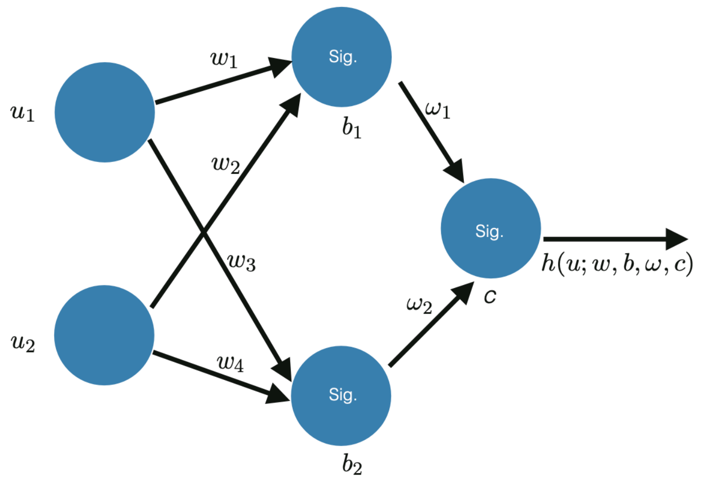
where x_i are the inputs to the neuron, w_i are the corresponding weights, b is a bias term, and f is a non-linear activation function.
Convolutional Layers (specific to CNNs): In CNNs, convolutional layers apply filters to the input, capturing spatial features:

$$(S * K)(i, j) = \sum_{u=1}^{m} \sum_{v=1}^{m} S(i + u, j + v) K(u, v),$$

where S is the input matrix, K is a kernel of size $m \times m$, and i, j index the output feature map.
Activation Functions: Activation functions such as ReLU (Rectified Linear Unit) introduce non-linear properties to the network:

$$f(x) = \text{máx}(0, x),$$

allowing the network to learn complex patterns in data.



A simple neural network

Learning and Optimization

Backpropagation: Neural networks adjust their weights through backpropagation, where errors between the predicted and actual outputs are propagated back through the network to update the weights.
Gradient Descent: This optimization method updates weights by moving in the direction that minimally increases the loss function:

$$\mathbf{w}_{new} = \mathbf{w}_{\text{old}} - \eta \nabla \mathcal{L}(\mathbf{w}),$$

where η is the learning rate and $\nabla \mathcal{L}(\mathbf{w})$ is the gradient of the loss function with respect to the weights.
We will see in detail the main stochastic optimization methods in next lecture.

Stochastic Optimization

An optimization problem seeks to find the minimum or maximum of a function subject to constraints, formally defined as:

$$\text{Minimize } f(x) \quad \text{subject to } x \in \mathcal{X},$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and $\mathcal{X} \subseteq \mathbb{R}^n$ denotes the feasible set.

8.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of stochastic optimization algorithms inspired by biological evolution. They operate on a population P of candidate solutions (individuals), evolving this population through iterations (generations) using mechanisms analogous to those found in natural selection and genetics.
An evolutionary algorithm can generally be described by the following steps:

1. **Initialization:** Generate an initial population $P^{(0)}$ of N individuals randomly. Each individual $x_i^{(0)}$ represents a potential solution to the optimization problem.

2. **Evaluation:** Compute the fitness $f(x_i)$ for each individual x_i in the population, where $f : \mathcal{X} \to \mathbb{R}$ is the fitness function that measures the quality of solutions.

3. **Selection:** Select individuals based on their fitness to form a mating pool. The selection probability $P_s(x_i)$ of an individual x_i often depends on its relative fitness to the population:

$$P_s(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}$$

4. **Crossover:** Apply the crossover operator to pairs of individuals in the mating pool to produce offspring, forming a new generation. A common method is the arithmetic crossover, where the offspring is a weighted average of the parents:

$$x_{\text{offspring}} = \alpha x_{\text{parent1}} + (1 - \alpha) x_{\text{parent2}}$$

   where x_{parent1} and x_{parent2} are the genetic vectors of the two selected parents, and α is a randomly chosen weight between 0 and 1.

5. **Mutation:** Apply the mutation operator with a small probability to new offspring to maintain genetic diversity within the population. Mutation randomly alters the genes of individuals, modeled as:

$$x_i' = x_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

   where ϵ represents a small random perturbation.

6. **Replacement:** Generate the next generation population by replacing some or all of the older individuals with new offspring, based on their fitness.

7. **Termination:** Repeat the evaluation through replacement steps until a termination criterion is met (e.g., a maximum number of generations or a satisfactory fitness level).

Evolutionary algorithms are characterized by their adaptability and robustness in various optimization tasks. They are capable of converging to a global optimum under certain conditions, making them suitable for complex solution spaces.

Theorem 8.1 (Schema Theorem and Global Convergence)**.** Under ideal conditions, such as infinite population sizes and generations, evolutionary algorithms can converge to a global optimum in continuous optimization problems. In genetic algorithms, patterns (schemata) with higher fitness than average tend to increase in frequency over generations. This is expressed as:

$$E(m_{H,t+1}) \geq m_{H,t}\frac{f_H}{\bar{f}}$$

where $m_{H,t}$ is the number of instances of schema H at generation t, f_H is the average fitness of those instances, and $\bar{f}$ is the average fitness of the population. This shows that favorable genetic patterns spread more, guiding the population towards better solutions.

Evolutionary algorithms exhibit robust performance across a broad spectrum of problems, benefiting from their ability to avoid local optima through stochastic behaviors such as mutations. The convergence rate can vary greatly and is influenced by factors such as the fitness landscape and algorithm parameters like mutation rates and crossover methods.

The complexity of evolutionary algorithms is primarily influenced by the population size, the dimensionality of the solution space, and the computational cost of fitness evaluations. The long-term behavior and stability of these algorithms can be modeled using dynamical systems theory and Markov chains, ensuring that performance does not deteriorate over time.

Genetic Algorithms (GA)

- Typically use binary strings or fixed-length vectors.
- Example: $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$, where $x_{ij} \in \{0, 1\}$.

Regarding mutation:

- Usually involves flipping bits in a binary string.
- Example: If $\mathbf{x}_i = [1, 0, 1]$, mutation might change it to $\mathbf{x}_i' = [1, 1, 1]$.

Crossover:

- Combines parts of two parent solutions to create offspring.

- Example: Single-point crossover:

$$\text{Parent 1: } \mathbf{x}_i = [1, 0, 1 \mid 0, 1, 1]$$

$$\text{Parent 2: } \mathbf{x}_j = [0, 1, 0 \mid 1, 0, 0]$$

$$\text{Offspring: } \mathbf{y}_k = [1, 0, 1 \mid 1, 0, 0]$$

Selection:

- Based on fitness, using methods like roulette wheel or tournament selection.
- Example: Probability of selection $P(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_j f(\mathbf{x}_j)}$.

Differential Evolution (DE)

Representation:

- Uses real-valued vectors.
- Example: $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$, where $x_{ij} \in \mathbb{R}$.

Mutation:

- Creates a donor vector by adding the weighted difference between two population vectors to a third vector.
- Example:

$$\mathbf{v}_i = \mathbf{x}_{r1} + F \cdot (\mathbf{x}_{r2} - \mathbf{x}_{r3})$$

  where $r1, r2, r3$ are distinct indices, and F is a scaling factor.

Crossover:

- Combines donor vector with target vector to create a trial vector.
- Example: Binomial crossover:

$$u_{ij} = \begin{cases} v_{ij} & \text{if } \text{rand}_j(0, 1) \leq CR \text{ or } j = j_{rand} \\ x_{ij} & \text{otherwise} \end{cases}$$

  where CR is the crossover probability.

Selection:

- Compares trial vector with target vector and selects the one with better fitness.
- Example:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i & \text{if } f(\mathbf{u}_i) \leq f(\mathbf{x}_i) \\ \mathbf{x}_i & \text{otherwise} \end{cases}$$

8.2 Stochastic gradient descent

Deterministic Optimization

For convex and differentiable functions, the gradient descent update rule is:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

ensuring convergence to the global minimum under appropriate conditions on the step size α_k.

Example (Gradient Descent on a Quadratic Function). Consider minimizing a simple quadratic function $f(x) = \frac{1}{2}x^2$. The gradient of this function is $\nabla f(x) = x$, and applying the gradient descent update rule with a fixed step size $\alpha_k = 0{,}1$, starting from $x_0 = 10$, we get:

$$x_1 = 10 - 0{,}1 \times 10 = 9,$$
$$x_2 = 9 - 0{,}1 \times 9 = 8{,}1,$$

and so forth. This sequence shows that x_k converges to 0 as k increases, illustrating the rate at which values approach the minimum.

Newton's Method: This is another powerful technique in deterministic optimization, particularly useful for functions that are twice differentiable. It improves upon gradient descent by considering the curvature of the function. The update formula for Newton's method is:

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k),$$

where $H^{-1}(x_k)$ is the inverse of the Hessian matrix at x_k. This method provides quadratic convergence near the optimum under suitable conditions, significantly faster than the linear convergence of gradient descent.

However, deterministic methods can be limited in practical scenarios where the objective function is noisy or only partially known, which leads us to Stochastic Optimization.

Stochastic optimization is necessary when the function f is noisy or evaluations are computationally expensive, requiring methods that can handle uncertainty and incomplete information. Updates in Stochastic Gradient Descent (SGD) are based on noisy estimates of the gradient:

$$\beta_{k+1} = \beta_k - \alpha_k \tilde{\nabla} f(\beta_k),$$

where α_k is the learning rate, and $\tilde{\nabla} f(\beta_k)$ represents a stochastic approximation of the gradient.

Monte Carlo Approximation

Monte Carlo methods estimate expectations by random sampling. To approximate the gradient $\nabla f(\beta_k)$ when the exact calculation is computationally expensive, we use:

$$\tilde{\nabla} f(\beta_k) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla f(\beta_i),$$

where β_i are independent and identically distributed (i.i.d.) samples drawn from the distribution of β, and N is the number of samples. This approach is particularly useful when the gradient $\nabla f(\beta)$ cannot be computed exactly or when it's too expensive to evaluate over the entire dataset.
Monte Carlo methods leverage the law of large numbers, which states that as the number of samples N increases, the average of the sampled gradients converges to the true gradient. This reduces the variance of the gradient estimate, making the update direction more reliable while maintaining computational efficiency.

ADAM Optimizer

ADAM (Adaptive Moment Estimation) is an extension of stochastic gradient descent that incorporates momentum and adaptive learning rates for each parameter. It combines the advantages of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp.
Key Features of ADAM:

- It stores an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t.
- It corrects these moments to counteract their initialization at the origin.

Update Rule: The parameters are updated as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla f(x_t), \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla f(x_t))^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ x_{t+1} &= x_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}. \end{aligned}$$

where:

- β_1 and β_2 are parameters controlling the exponential decay rates of these moving averages.
- α is the learning rate.
- ϵ is a small scalar used to prevent division by zero.

This update mechanism adjusts the learning rate for each parameter based on estimates of first and second moments of the gradients, enabling efficient convergence even for functions with noisy or sparse gradients.

Advanced Gradient Methods focus on modern techniques with momentum and adaptive rates, crucial for applications in deep learning. Distributed Optimization cover optimization over finite sums and distributed techniques, including the decentralized gradient descent model:

$$x_i^{k+1} = x_i^k - \alpha_k \left(\nabla f_i(x_i^k) + \sum_{j \in \mathcal{N}_i} w_{ij}(x_i^k - x_j^k) \right).$$

Mini-Batch Gradient Descent

Another common approach in SGD is to use a mini-batch of data points to approximate the gradient. Let B_k be the mini-batch at iteration k. The approximate gradient is:

$$\tilde{\nabla} f(\beta_k) = \frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(\beta_k),$$

where $f_i(\beta_k)$ is the objective function evaluated at sample i in the mini-batch, and $|B_k|$ denotes the number of samples in the mini-batch.

Why These Methods Work

- *Stochastic Nature*: Functions like $f(\beta)$ might represent expectations over a probability distribution. By sampling, we capture the variability in the data, and the average gradient of these samples provides an unbiased estimate of the true gradient.
- *Law of Large Numbers*: As the number of samples increases, the average of the sampled gradients converges to the true gradient, reducing the variance of the estimate.
- *Computational Efficiency*: Using subsets of data or random samples significantly reduces the computational cost per iteration compared to deterministic gradient descent.

Both Monte Carlo approximation and mini-batch gradient descent are essential for efficiently handling large-scale and noisy optimization problems, offering a balance between computational efficiency and gradient estimation accuracy.

Distributed Optimization

Distributed optimization involves performing optimization tasks across multiple processors or nodes to reduce the computational burden on a single processor and enhance scalability. Let $\mathcal{D}$ be the entire dataset, partitioned into M subsets $\mathcal{D}_m$ such that:

$$\mathcal{D} = \bigcup_{m=1}^{M} \mathcal{D}_m.$$

Each node m holds a subset $\mathcal{D}_m$ and performs local computations. The global optimization problem is:

$$\min_{x \in \mathbb{R}^n} \frac{1}{M} \sum_{m=1}^{M} F_m(x),$$

where $F_m(x) = \frac{1}{|\mathcal{D}_m|} \sum_{i \in \mathcal{D}_m} f_i(x)$.
In distributed gradient descent, each node computes its local gradient:

$$\tilde{\nabla} F_m(x_k) = \frac{1}{|\mathcal{D}_m|} \sum_{i \in \mathcal{D}_m} \nabla f_i(x_k),$$

and the gradients are aggregated to update the global parameter x:

$$x_{k+1} = x_k - \alpha_k \frac{1}{M} \sum_{m=1}^{M} \tilde{\nabla} F_m(x_k).$$

Decentralized Optimization

Decentralized optimization is performed in a networked system where data or computation is distributed among multiple agents or nodes, minimizing the need for centralized control. Each node m holds its own data $\mathcal{D}_m$ and communicates only with its neighbors. The optimization objective remains the same:

$$\min_{x \in \mathbb{R}^n} \frac{1}{M} \sum_{m=1}^{M} F_m(x).$$

One common method is decentralized gradient descent, where each node updates its parameters based on local data and sporadic communication with neighboring nodes. Let $\mathcal{N}(m)$ denote the set of neighbors of node m. Each node performs the following updates:
1. Compute local gradient:

$$\tilde{\nabla} F_m(x_k) = \frac{1}{|\mathcal{D}_m|} \sum_{i \in \mathcal{D}_m} \nabla f_i(x_k).$$

2. Communicate with neighbors and update local parameter:

$$x_m^{(k+1)} = \sum_{n \in \mathcal{N}(m)} w_{mn} x_n^{(k)} - \alpha_k \tilde{\nabla} F_m(x_m^{(k)}),$$

where w_{mn} are weights representing the influence of neighboring nodes.
This method ensures that each node converges to a common solution while maintaining a decentralized structure, which is crucial for large-scale and privacy-sensitive applications.
Both distributed and decentralized optimization techniques are essential for efficiently solving large-scale optimization problems by leveraging the computational power of multiple nodes and minimizing the drawbacks of centralized control.

8.3 Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is an iterative method designed to find the maximum likelihood estimates of parameters in probabilistic models, particularly when data are incomplete or have missing values.

Algorithm Overview

- **Expectation Step (E-step):** Calculate the expected value of the log-likelihood function, with respect to the conditional distribution of the latent variables given the observed data and the current parameter estimates.

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_{Z|X,\theta^{(t)}}[\log L(\theta; X, Z)]$$

  where $\theta^{(t)}$ is the parameter estimate at iteration t, X represents the observed data, Z represents the latent variables, and $L(\theta; X, Z)$ is the complete-data likelihood function.

- **Maximization Step (M-step):** Update the parameters to maximize the expected log-likelihood found in the E-step:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)})$$

The EM algorithm alternates between these two steps until convergence, typically achieving a local maximum of the likelihood function.

Example (Biomedical Imaging). Consider the task of image reconstruction in positron emission tomography (PET), where the E-step estimates the distribution of decay events given the observed data, and the M-step updates the image to maximize the likelihood of the observed data given the decay events.

Monte Carlo Expectation Maximization (MCEM)

The Monte Carlo Expectation Maximization (MCEM) algorithm extends the EM framework to situations where the E-step cannot be computed analytically. Instead, it relies on Monte Carlo methods to approximate the expectation.

MCEM Procedure

1. **Simulation Step:** Generate samples from the distribution of the latent variables conditioned on the observed data and current parameter estimates.

2. **Monte Carlo E-step:** Approximate the expectation of the complete-data log-likelihood using the sampled values:

$$\tilde{Q}(\theta|\theta^{(t)}) = \frac{1}{N}\sum_{n=1}^{N} \log L(\theta; X, Z^{(n)})$$

   where $Z^{(n)}$ are the samples of latent variables.

3. **M-step:** Maximize the estimated Q-function:

$$\theta^{(t+1)} = \arg\max_{\theta} \tilde{Q}(\theta|\theta^{(t)})$$

Importance Sampling in MCEM

Importance sampling is used in the MCEM to improve the efficiency of the Monte Carlo approximation, especially when direct sampling from the posterior distribution is challenging.

$$\hat{Q}(\theta|\theta^{(t)}) = \sum_{n=1}^{N} w_n \log L(\theta; X, Z^{(n)}) \tag{8.1}$$

where w_n are the importance weights correcting for the bias in the sampling distribution, and $Z^{(n)}$ are the sampled latent variables.

Example (Epidemiological Modeling). In modeling the spread of an infectious disease, use MCEM to estimate the transmission rates and recovery rates from incomplete data, where importance sampling addresses the variability in individual susceptibility and recovery rates.

8.4 Markov Chain Monte Carlo for Optimization

Markov Chain Monte Carlo (MCMC) methods are primarily known for their application in Bayesian statistics for sampling from complex posterior distributions. However, these techniques can also be effectively used for solving optimization problems, especially those involving high-dimensional spaces or complex, multi-modal landscapes.
The Markov chain generated by the Metropolis-Hastings algorithm converges to a stationary distribution that is proportional to $f(x)$. For optimization purposes, $f(x)$ can be chosen as a function that peaks at the global optimum, such as $e^{\beta f(x)}$, where β is a parameter controlling the sharpness of the peak.
In machine learning, MCMC methods can be used to optimize likelihood functions or posterior distributions in settings where direct optimization is challenging due to the complexity of the model or the presence of multiple local optima.

No Free Lunch Theorems

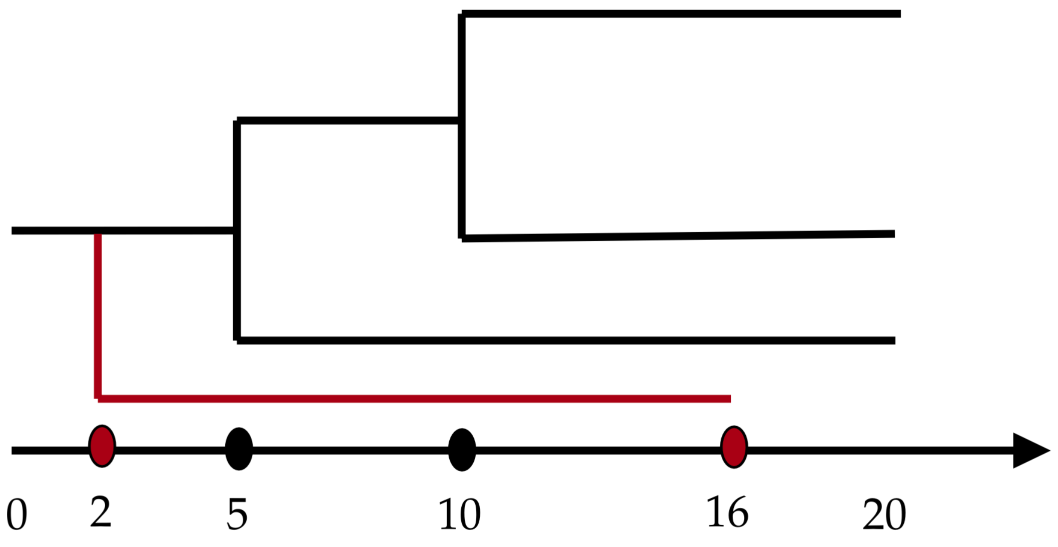
These theorems articulate that no optimization algorithm outperforms others when averaged across all possible problems.

Theorem 8.2 (No Free Lunch Theorems)**.** The performance of any optimization algorithm averaged over all possible problems yields the same result, suggesting that no universally optimal algorithm exists.

Project

Part 1: Computational evolutionary processes

Evolutionary dynamics explore the underlying processes that shape the development and diversity of systems over time across a variety of fields. These dynamics are commonly modeled using stochastic processes, such as birth-death processes, where 'births' and 'deaths' metaphorically denote the emergence and disappearance of elements like languages, species, innovations, or financial products. Such models help illustrate the evolution of these diverse systems through time.
Consider the following simple evolutionary tree, where the x-axis represents time in millions of years:



Modeling

Consider a model that addresses the following rates:

- **Innovation rate** (λ): This rate is diversity-dependent and formulated as:

$$\lambda_t = \text{máx}\{(\lambda_0 - \beta_N N_t)\,,0\}$$

  where λ_0 is the baseline innovation rate under ideal conditions (no competition) and N_t is the number of branches at time t. The function máx ensures that the innovation rate does not drop below zero.

- **Extinction rate** (μ): The extinction rate is assumed to be a constant value, μ_0, reflecting a baseline risk of extinction that does not vary with time:

$$\mu = \text{máx}\{\mu_0, 0\}$$

By adjusting the innovation rate based on diversity, the model accounts for competition and niche filling, which are pivotal in understanding evolutionary patterns. The constant extinction rate simplifies the complexity of the model while focusing on the dynamics of innovation.

1. Assuming that this process was generated under rates $\lambda_0 = 0{,}8$, $\beta_N = -0{,}075$ and $\mu_0 = 0{,}1$, calculate the probability of the evolutionary process described in the tree of the picture.

2. Now consider the case where the fossil record is missing. Assume that we know there is only one extinction and formulate the probability of the extant tree.

3. Use a Monte Carlo method, together with importance sampling, to estimate the integral above. To do that:

   a) Use the uniform sampler. That is, sample the innovation time, the extinction time, and the allocation of the missing branches from uniform distributions.

   b) We can obtain a numerical approximation by using the Monte-Carlo approach considering

$$\begin{aligned} f(x_{obs}|\theta) &= \int_{x\in\mathcal{X}(x_{obs})} f(x|\theta)dx \\ &= \int_{x\in\mathcal{X}(x_{obs})} \frac{f(x|\theta)}{f_m(x|\theta, x_{obs})} f_m(x|\theta, x_{obs})dx \\ &\approx \frac{1}{M} \sum_{x_i \sim f_m(x|\theta, x_{obs})} \frac{f(x_i|\theta)}{f_m(x_i|\theta, x_{obs})} \end{aligned} \tag{9.1}$$

   for M the Monte-Carlo sample size and f_m is your sampler of the missing part of the full tree given an extant tree x_{obs}.

4. Now assume we do not know how many extincted branches are missing. Design an implement a stochastic method to calculate the probability of the number of missing branches.

5. Propose another sampler, designed by you, and compare the results.

6. Extend the model to non-constant extinction rate, perform simulations with both models and compare the results. Calculate the probabilities of points 3. and 4. for the non-constant extinction rate model.

Part 2: Simulation and Learning

In this part, you will use stochastic methods for simulating and training systems that learn. You have two alternatives to choose from:

- Augmenting trees and then training a Generalized Additive Model (GAM)
- Simulating full trees and then training a Neural Network (NN)

Choose one of the following approaches:

2.1 Augmenting Trees and Training a GAM

In this approach, you will extend the evolutionary trees generated in Part 1 with additional simulated data to train a Generalized Additive Model (GAM). The GAM will be used to predict the likelihood function as a function of the model parameters. Follow these steps:

1. Simulate additional evolutionary trees by varying the parameters λ_0, β_N, and μ_0 according to the diversity dependence model described previously:
    - λ_0: Baseline innovation rate
    - β_N: Diversity dependence parameter
    - μ_0: Baseline extinction rate
2. Use the simulated trees to augment the dataset.
3. Estimate the joint probability or likelihood function for different sets of parameters $(\lambda_0, \beta_N, \mu_0)$.
4. Train a GAM using the augmented dataset to predict the likelihood function based on the tree characteristics and parameters.
5. Implement Differential Evolution (DE) or Stochastic Gradient Descent (SGD) to optimize the GAM and retrieve the proper parameters of the tree.
6. Evaluate the performance of the GAM by comparing its predictions to the known rates used in the simulations.

2.2 Simulating Full Trees and Training a NN

In this approach, you will simulate entire evolutionary trees and use them to train a Neural Network (NN). Follow these steps:

1. Simulate a diverse set of full evolutionary trees with varying parameters $(\lambda_0, \beta_N, \mu_0)$.
2. Encode the trees in a suitable format for neural network input.
3. Split the dataset into training, validation, and test sets.
4. Design and train a Neural Network to predict the parameters $(\lambda_0, \beta_N, \mu_0)$ from the encoded trees.
5. Evaluate the NN's performance on the test set.

Comparative Analysis for Advanced Students

For advanced students who want to explore both approaches, perform a comparative analysis:

1. Compare the predictive accuracy of the GAM and NN models.
2. Discuss the computational efficiency and scalability of each method.
3. Analyze the strengths and weaknesses of each approach.

Bonus: A simple importance sampler

To sample trees we propose a tree augmentation algorithm that samples independently the three components of the tree: event types, event times and species allocations. The algorithm is shown in Figure 9.1.

***S*tep 1. Generate event times and number of extinctions.** The number of extinct species d and $2d$ missing event times, i.e., speciations and extinctions of these d missing species are sampled uniformly in the following manner:

1. Sample the number of missing species d uniformly from the discrete space $\{0, \dots, M^e\}$ where M^e is a predefined ceiling, such that the probability of more than M^e extinctions is extremely unlikely.

2. Sample $2d$ branching times uniformly from the continuous space $(0, T]$ and then sort them.

The probability of sampling a set of $2d$ unobserved event times $t^e = (t^e_1, \dots, t^e_{2d})$ for a tree of dimension d is

$$g_{\text{event times}}(d, t^e) = \frac{1}{M^e + 1} \left(\frac{1}{T}\right)^{2d} (2d)!$$

Note that this scheme samples the dimension of the tree uniformly, but the size of the space of trees grows in a factorial way with the dimension of the tree. This means that the sample size required to obtain a robust Monte Carlo approximation of the integral must be large. This is a limitation of this importance sampler, and hence it is only reliable when many extinctions are unlikely.

***S*tep 2. Generate event types** We simulate a binary event chain $\tau^e = (\tau^e_1, \dots, \tau^e_{2d})$ assigning either S (speciation) or E (extinction) to each event time. This chain is subject to the rule that the number of Es up to any point in the chain should be less than or equal to the number of Ss in the chain up to that point. The set of allowed chains is known in the mathematical literature as the set of Dyck words and several methods for sampling Dyck words have been developed. Furthermore, given a number of events $2d$, the number of possible Dyck words is known as the Catalan number,

$$C_d = \binom{2d}{d} \frac{1}{d+1}.$$

By uniformly sampling a Dyck word τ^e of length $2d$, the probability of a specific event sequence is given by $g_{\text{events}}(\tau^e) = 1/C_d$.

***S*tep 3. Species allocation** Given the missing event times and missing event types we can perform the tree allocations by sampling a parent species of each missing speciation and by defining which species, i.e., the parent species or the inserted “new species”, becomes extinct at the extinction event. To sample uniformly we just need to count the number of possible trees in agreement with the event times $t^e = (t^e_1, \dots, t^e_{2d})$ and event types. This number, $n(\tau^e_{2d}, t^e_{2d})$, can be calculated by starting with $n(\tau^e_0, t^e_0) = 1$ and applying the following rules when going from root to tips in the phylogenetic tree:

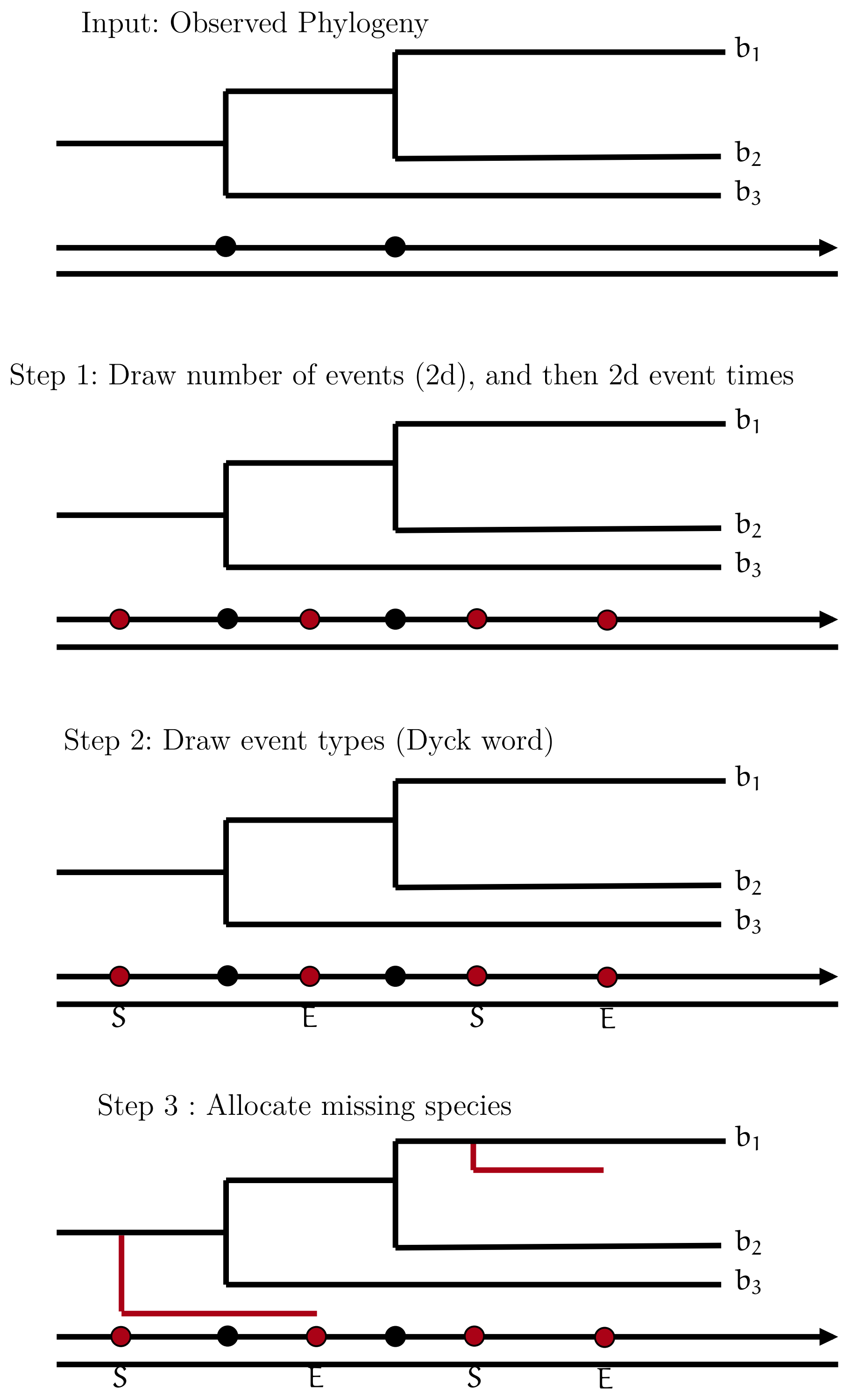

Input: Observed Phylogeny
b_1
b_2
b_3
Step 1: Draw number of events (2d), and then 2d event times
b_1
b_2
b_3
Step 2: Draw event types (Dyck word)
b_1
b_2
b_3
S
E
S
E
Step 3 : Allocate missing species
b_1
b_2
b_3
S
E
S
E

Figura 9.1: The three components of our phylogenetic tree augmentation algorithm.

- For each unobserved speciation event at t_i^e, i.e., $\tau_i^e = S$, update $n(\tau_i^e, t_i^e)$ in the following way,
$$n(\tau_i^e = S, t_i^e) = n(\tau_{i-1}^e, t_{i-1}^e) \times \left(2N_{t_i^-}^o + N_{t_i^-}^e\right),$$
where N_{t-}^o is the number of observed branches just before t and N_{t-}^e is the number of unobserved branches just before t.

- For each unobserved extinction event at t_i^e, i.e., $\tau_i^e = E$, update $n(\tau_i^e, t_i^e)$ in the following way,
$$n(\tau_i^e = E, t_i^e) = n(\tau_{i-1}^e, t_{i-1}^e) \times N_{t_i^-}^e.$$

As we sample uniformly, the probability for each possible allocation a^e of the d missing species at the missing event times t^e with Dyck word τ^e in the tree of extant species x_{obs} is then given by $g_{\text{allocation}}(a^e) = \frac{1}{n(\tau_{2d}^e, t_{2d}^e)}$.

Sampling probability of a uniformly augmented tree The uniform sampling probability of the augmented tree $x_{\text{unobs}} = (d, t^e, \tau^e, a^e)$ is then given by

$$g(x_{\text{unobs}}|x_{\text{obs}}, \theta) = \frac{1}{M^e + 1} \left(\frac{1}{T}\right)^{2d} (2d)! \frac{1}{C_d} \frac{1}{n(\tau_{2d}^e, t_{2d}^e)} \tag{9.2}$$

From this equation we can see how the dimension of the tree space plays an important role. For this reason, the uniform importance sampler becomes less efficient when many extinctions are likely. On the other hand, the uniform sampling scheme allows for easy implementation and quick computation, thereby making it suitable as a default sampler.