Lecture Notes

Stochastic Simulation and Monte Carlo Method

Davoud Mirzaei

Uppsala University

November 1, 2024 (2nd Edition)

Contents

These lecture notes are intended to cover some introductory topics in stochastic simulation for scientific computing courses offered by the IT department at Uppsala University, as taught by the author. Basic concepts in probability theory are provided in the Appendix A, which you may review before starting the upcoming sections or refer to as needed throughout the text. Some parts of our presentation here follow [**DeGroot-Schervish:2007**], [**Ross:2002**] and [**Rubinnstein-Kroese:2017**].

1 Deterministic vs. Stochastic

In the field of scientific computing, modeling and simulation are fundamental tools used to understand natural phenomena. Two main approaches are commonly employed for modeling and simulation: **deterministic** and **stochastic**. While both approaches aim to predict the behavior of systems, they differ in how they handle uncertainty and randomness.

1.1 Deterministic models and methods

A deterministic model is one in which the behavior of the system is entirely predictable and reproducible. Given a specific set of initial conditions and parameters, the outcome will always be the same, i.e. the future behavior can be predicted with complete certainty. Such models are often governed by mathematical equations (such as differential equations or algebraic relations) that describe precise relationships between different quantities. For example, consider Newton's laws of motion, which describe how objects move in response to forces. These laws are deterministic because, given the initial conditions (such as the position, velocity, and force acting on an object) the future motion of the object can be calculated exactly. All mathematical models expressed as ordinary and partial differential equations (ODEs and PDEs) are examples of deterministic models.

Deterministic methods are employed to solve deterministic models[1]. Methods such as Euler's method for solving ODEs, the trapezoidal rule for integration, and the Newton-Raphson method for solving non-linear equations are basic examples of deterministic methods. When it comes to more complicated models such as PDEs, the finite difference method (FDM), the finite element method (FEM), and the finite volume method (FVM) are examples of commonly used deterministic methods for solving such deterministic models.

As a simple example, consider the decay of a radioactive material. This process can be described by a first-order differential equation, which relates the rate of decay to the amount of radioactive material present at a given time. The governing equation has the form

$$\frac{dy}{dt} = -\lambda y(t)$$

where $y(t)$ represents the amount of radioactive material at time t, and λ is the decay constant.

[1]It is sometimes possible to use a stochastic method to solve a deterministic model. For example, a Monte Carlo method can be used to approximate the volume of an object, which is equivalent to solving a multi-dimensional integral.

The initial amount of material is given by $y(0) = y_0$. This model is deterministic because, given the initial amount of radioactive material y_0 and the decay constant λ, the future amount of material can be calculated precisely at any time. Indeed, the solution to this equation is

$$y(t) = y_0 e^{-\lambda t}.$$

This solution describes the exponential decay of the material over time. No matter how many times we perform this calculation, we will always obtain the same result for a given set of input parameters and initial conditions.

However, solving differential equations analytically is not always possible or practical. In such cases, numerical methods are used to approximate the solutions. One common solver is the Runge-Kutta method. An adaptive version of this method has been implemented in Python library `scipy.integrate.solve_ivp`. Here, we call this library and compute a numerical solution to the radioactive decay equation for $\lambda = 0.5$ and $y_0 = 10^3$.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
lam, y0, FinalTime  = 0.5, 1000, 10   # rate, initial value, final time
def ODEfun(t,y):
    yprime = -lam*y
    return yprime
teval = np.linspace(0, FinalTime, 500)
sol = solve_ivp(ODEfun, [0,FinalTime], y0, t_eval = teval)
plt.figure(figsize = (6, 4))
plt.plot(sol.t,sol.y[0],linestyle = 'solid', color='blue')
plt.xlabel('$t$ (seconds)');
plt.ylabel('Amount of radioactive material, $y(t)$')
plt.title('Deterministic solution, radioactive decay: $y_0=1000,\lambda=0.5$')
```

The plot is given on the left-hand side of Figure 1.

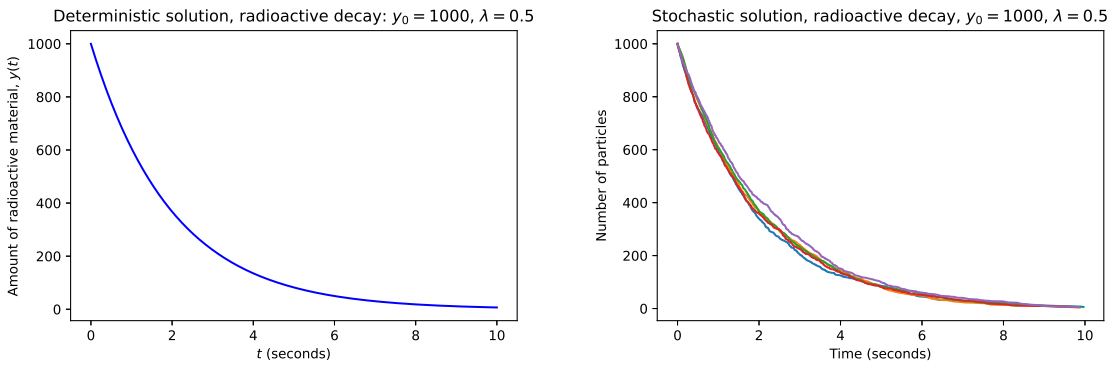


Figure 1: Numerical solution using the deterministic ODE-solver `RK45` from Python library `scipy.integrate.solve_ivp` (left), and using the stochastic solver SSA (right)

As we observe, similar to the analytical solution, the numerical scheme yields a deterministic outcome for the given input values. Randomness does not play a role here, and all results are fixed. For instance, the amount of material at time $t = 4$ is a precise value, $y_4 = 135.3353$ (rounded to four decimal places).

1.2 Stochastic models and methods

Deterministic models are not always an accurate reflection of reality. For instance, in the above example the exact time at which an individual atom will decay is random. Similarly, in population dynamics, deterministic models cannot account for the random events such as environmental changes or disease outbreaks. Stochastic models incorporate randomness and uncertainty to have a better reflection of the behavior of real-world systems. In a stochastic model, the same set of initial conditions can lead to different outcomes. Instead of predicting a single and precise result, these models describe a range of possible outcomes, each associated with a certain probability. These models are particularly useful in areas such as biology, finance, and quantum mechanics where small-scale fluctuations or random events play a significant role in determining the overall behavior of the system.

Let us revisit the example of radioactive decay. In the deterministic model, we used a differential equation to predict the amount of radioactive material at a given time. However, this model assumes that the decay process occurs continuously and deterministically, which is not the case in reality. In fact, radioactive decay is a random process, and the time at which each atom decays is uncertain. In a stochastic model, we describe the system as a series of random events. Specifically, we model the decay process as a reaction:

$$y \xrightarrow{\lambda} z$$

where y represents the number of radioactive molecules, z represents the decay products, and λ is the *propensity* of decay. The difference between this model and the deterministic model is that we no longer assume a continuous, predictable decay process. Instead, we model the decay as a random event that occurs with a certain probability, and the waiting time for the next decay is also a random variable. This is one of the key features of stochastic models that they contain random variables and *probability distributions* to describe the likelihood of different outcomes.

To simulate this stochastic model, we can use a method known as *Gillespie's Algorithm*, or the *Stochastic Simulation Algorithm (SSA)*. See section 7 for details. Each time we run the simulation, we will obtain a different result which reflects the randomness of the decay process.

See the right panel in Figure 1 for five simulation outcomes, with the respective values of y at $t = 4$ being 122, 228, 134, 135, and 139 for each simulation. In this example, if we run the simulation many times and compute the average behavior of the system, we will find that the result closely approximates the deterministic solution. However, this is not always the case. When we model a phenomenon using both a stochastic model and a deterministic model (as

we did with radioactive decay), the mean of stochastic solutions does not always approximate the deterministic solution. In some cases, while the deterministic solution converges to an equilibrium state, random noise and fluctuations may force all stochastic solutions to deviate from the equilibrium.

1.3 Which one?

The choice between a deterministic and a stochastic model depends on the nature of the system being studied and the goals of the simulation. Deterministic models are ideal for systems that behave predictably, where small-scale fluctuations have little impact on the overall behavior of the system. These models are computationally efficient and provide precise and repeatable results. They are well-suited for systems where accuracy is important. However, deterministic models can be limited in their ability to describe some real-world systems that are subject to randomness and uncertainty. In such cases, stochastic models provide a more realistic description of the system behavior. By incorporating randomness, stochastic models can capture the variability and uncertainty.

One important consideration when choosing between a deterministic and a stochastic model is the size of the system. In large systems (e.g. a model with many particles), random fluctuations tend to average out, and the overall behavior of the system can be described accurately by a deterministic model. However, in small systems (e.g. a model with a few number of species), random events can have a significant impact on the system behavior, and make a stochastic model more appropriate.

Another consideration is the computational cost of the simulation. Deterministic models are typically more computationally efficient than stochastic models, as they require fewer simulations to obtain a precise result. Stochastic models, on the other hand, require many simulations to accurately estimate the probability distribution of different outcomes which makes them more computationally intensive.

2 Monte Carlo method – I

We begin by introducing the Monte Carlo (MC) algorithm as a stochastic method. This method was invented by *John von Neumann* and *Stanislaw Ulam* during World War II to improve decision making under uncertain conditions. The name *Monte Carlo* is motivated by the randomness similar to games in the Monte Carlo casino.

In this section, we try to present the basic concepts behind the MC method. Since MC and other stochastic methods rely on random points, the next section is devoted to some techniques for generating random variables. Following that, we will revisit and further consider the MC algorithm in section 4.

2.1 Let's play a game

We play the Snakes-and-Ladders game using a 6-sided dice. The game board is shown in Figure 2. The game rules are:

- Start from space 1, roll the dice and move forward the number of spaces shown on the dice.
- If you land at the base of a ladder move up to the top of the ladder.
- If you land at the head of a snake slide down to the bottom of the snake.
- To finish the game you need to roll the exact number to get you to the last space 100. For example, if you are at space 99 then you should toss the dice until getting 1 to finish.

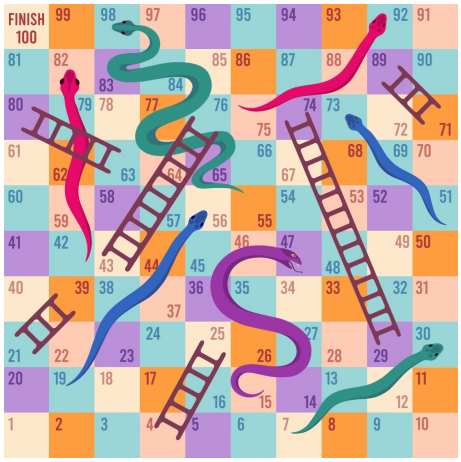


Figure 2: A Snakes-and-Ladders game board (image from `www.vectorstock.com`).

Let X represent the number of dice rolls required to reach the finish space. Our goal is to determine the expected number of rolls required to finish, or in the mathematics language, to compute the expectation of X, denoted as $\mathbb{E}(X)$.

To answer this, we can ideally find a mathematical expression for the *probability density function (pdf)* of the random variable X and then compute $\mathbb{E}(X)$ using the formula for expectation provided in Definition A.4 in the Appendix. However, finding a closed-form expression for the pdf of X is either impossible or extremely difficult. Instead, we will go for a simple numerical solution.

Assume that one individual plays the game and finishes it after, say, $x_1 = 51$ rolls. This single *observation* (*realization*) is insufficient to conclude that the game will always end after 51 rolls. However, if N individuals play the game, we can collect N observations $x_1, x_2, \ldots, x_N$ of X. The approximate expected number of rolls can then be estimated by calculating the average (mean) of these N observations:

$$\mathbb{E}(X) \approx \frac{1}{N}(x_1 + x_2 + \cdots + x_N).$$

This is a Monte Carlo solution: perform the experiment many times and compute the average

of the results. As the number of observations N increases, our estimation of the expected number becomes more accurate.

In practice, rather than playing the game manually, we can write a program to simulate it. To simulate the rolling of the dice, we generate random numbers from *discrete uniform distribution* $\mathcal{DU}\{1, 2, \ldots, 6\}$, where each outcome has a probability of 1/6. Refer to the Appendix for definitions and notations of well-known probability distributions. Additionally, the code must account for the effect of snakes and ladders, which can move the player forward or backward on the board.

A simulation performed with $N = 10,000$ observations shows that the expected number of rolls, $\mathbb{E}(X)$, is approximately 44. This means that in average people finish this game after approximately 44 rolls. Other possible questions we can answer via the above MC implementation are:

- What is the probability of finishing the game by exactly 30 rolls? To answer this, we look at the MC observations, count the number of 30s, and divide it by the total number of observations:
$$\mathbb{P}(X = 30) \approx \frac{\#30s}{N} \approx 0.02$$
- What is the probability of finishing the game by at most 30 rolls?
$$\mathbb{P}(X \leqslant 30) \approx \frac{\#30s + \#29s + \cdots + \#1s}{N} \approx 0.34$$

This approach allows us to approximate the probability based on the frequency of occurrences in the Monte Carlo simulation.

2.2 A general structure

There is no single, universally accepted definition of the Monte Carlo method. Its formulation varies based on the underlying mathematical model and the specific type of solution being sought. However, a rather general structure is outlined in Algorithm 1 below.

Algorithm 1 A general structure of Monte Carlo

Require: Number of observations N
  for k form 1 to N **do**
    Perform one *stochastic simulation*
    Set $result[k]$ = result of the simulation
  end for
  $FinalResult$ = mean($result$) or other statistical calculations

The *stochastic simulation* can differ depending on the problem. It could be an observation of a random variable or an observation of a stochastic process. In the next section, we will see how to generate random variables from various probability distributions. Then in section 5 we study the random processes.

3 Random variable generation

As we pointed out, a typical stochastic simulation requires a set of random numbers, random variables, or a series of stochastic processes. In this and the next sections we deal with the computer generation of such entities.

We start with generating a uniform random point. Today's random numbers are generated by simple computer algorithms instead of physical devices such as coin flipping, dice rolling, roulette spinning, and card shuffling, or even modern physical generation methods such as those based on the universal background radiation or the noise of a PC chip. Random number generation algorithms are usually fast, require little storage space, and can readily reproduce a given sequence of random numbers. Although such sequences are generated by a deterministic algorithm, they fulfill main statistical properties of true random sequences. For this reason the generated numbers are sometimes called *pseudorandom* numbers. Here, we do not pursue the details of algorithms for generating uniform random numbers and just use the following Python function which uses the `uniform` function form the `numpy.random` library.

```
U =  np.random.uniform(a, b, size = N)
```

This generates N uniform random points in interval $[a, b]$. Another possible command is

```
U =  np.random.rand(N)
U = a + (b-a)*U
```

which first generates N uniform random points in standard interval $[0, 1]$ and then transfers them into interval $[a, b]$ using the linear map $x \mapsto a + (b - a)x$.

Now, we review some general methods for generating one-dimensional random variables from a prescribed distribution.

3.1 Inverse transform method

Let X be a random variable with pdf f and cdf F. If F is continuous and increasing then F^{-1} has the usual definition

$$F^{-1}(y) := \{x : F(x) = y\}.$$

To cover all cases including discrete and nondecreasing cdf functions, the inverse function F^{-1} can be defined as

$$F^{-1}(y) = \inf\{x : F(x) \geqslant y\}, \quad 0 \leqslant y \leqslant 1. \tag{3.1}$$

See Figure 3 for an illustration.

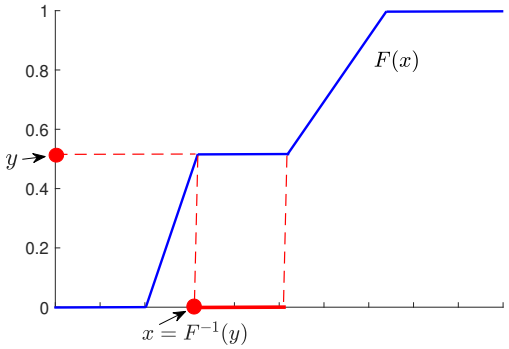


Figure 3: The inverse of a non-decreasing function

Now we have the following useful result.

Theorem 3.1. If $U \sim \mathcal{U}(0,1)$ then $X = F^{-1}(U) \sim f$

Proof. Since F is invertible and $\mathbb{P}(U \leqslant u) = u$, we have

$$\mathbb{P}(X \leqslant x) = \mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x),$$

which completes the proof. ■

Theorem 3.1 proposes a simple algorithm to generate a random variable X with cdf F (or pdf f): Generate $U \sim \mathcal{U}(0,1)$ and set $X = F^{-1}(U)$. An illustration is given in Figure 4, where the uniform variable U on the y-axis is transferred to f-distributed variable X on the x-axis via the cdf F.

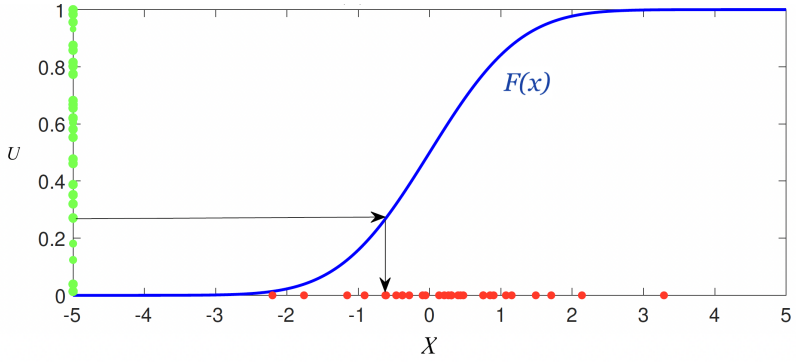


Figure 4: The inverse transform method

Example 3.1. To generate a random point from pdf

$$f(x) = \begin{cases} 2x, & x \in [0,1] \\ 0, & \text{otherwise}, \end{cases}$$

first we obtain its corresponding cfd

$$F(x) = \begin{cases} 0, & x \in (-\infty, 0), \\ x^2, & x \in [0,1] \\ 1, & \text{otherwise}, \end{cases}$$

then we generate a uniform variable U and finally we set $X = F^{-1}(U) = \sqrt{U}$.

Sampling from exponential distribution

If $X \sim \mathcal{E}xp(\lambda)$, then its pdf f is given by $f(x) = \lambda \mathrm{e}^{-\lambda x}$ and its cdf F by

$$F(x) = \int_0^x \lambda \mathrm{e}^{-\lambda y} \mathrm{d}y = 1 - \mathrm{e}^{-\lambda x}, \quad x \geqslant 0.$$

The inverse of F is $F^{-1}(x) = -\frac{1}{\lambda}\ln(1-x)$. Thus, to sample from the exponential distribution, we assume $U \sim \mathcal{U}(0,1)$ and set

$$X = -\frac{1}{\lambda}\ln(U) \sim \mathcal{E}xp(\lambda). \tag{3.2}$$

Keep in mind that $U \sim \mathcal{U}(0,1)$ implies $1 - U \sim \mathcal{U}(0,1)$.

Here we write a Python function to generate N random variable with exponential distribution.

```
def RandExp(lam,N):
    # lam: distribution parameter, N: number of requested samples
    U =  np.random.rand(N)  # generate N uniform numbers in [0,1)
    X = -1/lam*np.log(1-U)  # use inverse transform to generate X
    return X
```

The following code snippet plots the histogram of $N = 500$ generated points with parameter $\lambda = 0.5$.

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize = (5,3))
lam, N = 0.5, 500
X = RandExp(lam,N)
plt.hist(X, bins = 30, histtype = 'bar', color = 'red', density = 'true')
x = np.linspace(0,15,200)
f = lam*np.exp(-lam*x)
plt.plot(x,f,linestyle = '-', color = 'blue')
plt.title('Histogram of $X$ and the pdf $f(x)$')
plt.xlabel('$X$')
plt.ylabel('Frequency %')
```

In Figure 5 the histograms for $\lambda = 0.5$ with $N = 500$ and $N = 5000$ are shown. For comparison the pdf of the exponential distribution is also plotted in the figures.

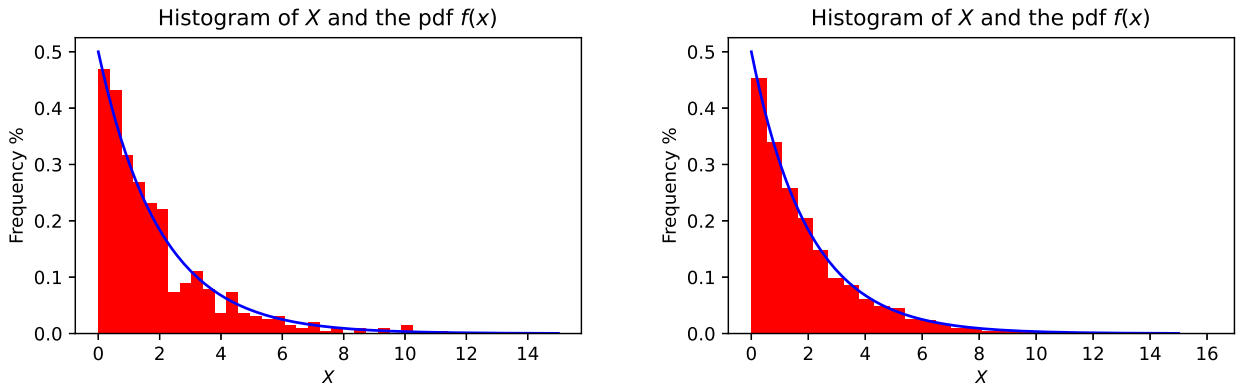


Figure 5: Histograms of generated random variables with exponential distribution $\mathcal{E}xp(0.5)$ using the inverse transform method. $N = 500$ (left), $N = 5000$ (right)

Sampling from normal distribution

In the Appendix A.8 we have shown how a change of variables can help to generate random points from the normal distribution using the inverse transform method. However, in our codes in the sequel, we use some built-in functions in Python. We either use

```
U =  np.random.normal(mu, sigma2, N)
```

which generates N normal random points with mean `mu` and variance `sigma2`, or use

```
U =  np.random.randn(N)
U = mu + sigma*U
```

which first generates N standard normal points (from $\mathcal{N}(0,1)$) and then transfers them into a new set of points with distribution $\mathcal{N}(\mu, \sigma^2)$ using the fact that if $Z \sim \mathcal{N}(0,1)$ then $X = \mu + \sigma Z \sim \mathcal{N}(\mu, \sigma^2)$.

Sampling from discrete distributions

So far, we have only sampled from continuous distributions. However, the inverse transform method can easily be applied to sample from discrete distributions as well. Let X be a discrete distribution with

$$\mathbb{P}(X = x_k) = p_k, \quad k = 1, 2, \ldots, m \quad \sum_{k=1}^{m} p_k = 1.$$

Without lose of generality, let $x_1 < x_2 < \cdots < x_m$. The cdf of X is given by

$$F(x) = \sum_{k: x_k \leqslant x} p_k.$$

The plot of F looks like a step-wise function and is shown in Figure 6.

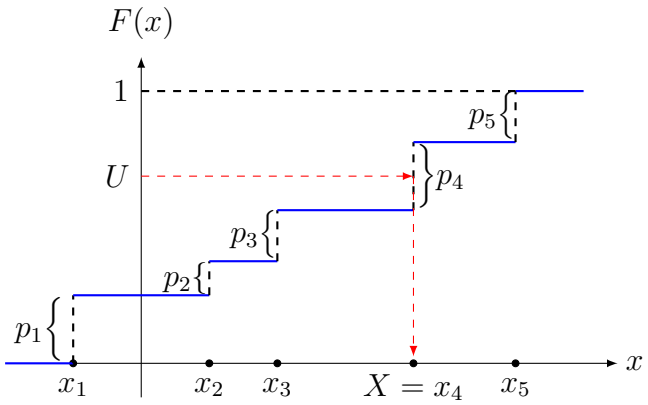


Figure 6: Inverse transform method for a discrete random variable.

To sample from such discrete random variable, according to definition of F^{-1} in (3.1), first we generate a uniform distribution $U \sim \mathcal{U}(0,1)$ and then we find the smallest positive integer k such that $U \leqslant F(x_k)$. Finally $X = x_k$ is reported. Equivalently

$$X = \begin{cases} x_1, & 0 < U \leqslant p_1 \\ x_2, & p_1 < U \leqslant p_1 + p_2 \\ x_3, & p_1 + p_2 < U \leqslant p_1 + p_2 + p_3 \\ \vdots & \vdots \end{cases}$$

A Python function is given below. The inputs are the sorted vector `x` containing the states x_k, the corresponding probability vector `p`, and `N` the number of samples requested. The output is the vector `X` containing `N` random points distributed with $\mathcal{DD}\{[x_1, \ldots, x_m], [p_1, \ldots, p_m]\}$.

```
def RandDisct(x, p, N):
    # x: sorted states, p: probabilities, N: number of requested samples
    cdf = np.cumsum(np.array(p))  # compute the cumulative vector
    U = np.random.rand(N)         # generate N uniform numbers in [0,1)
    idx = np.searchsorted(cdf, U) # search U values in cdf intervals
    X = np.array(x)[idx]
    return X
```

Example 3.2. Using the code snippet below, we roll a dice 10 times and report the result.

```
x = [1,2,3,4,5,6]
p = [1/6,1/6,1/6,1/6,1/6,1/6]
X = RandDisct(x, p, 10)
print('dice rolls = ', X)
```

The result of a run is

```
dice rolls =  [5 2 4 4 4 5 6 2 1 3]
```

To sample from the Bernoulli distribution $\mathcal{B}er(a)$ for $a \in [0,1]$ we can call the `RandDisc` function with `x = [0,1]` and `p = [1-a,a]`. We can also use the following independent function to sample a vector of Bernoulli variables with probability $p \in [0,1]$. This function generates a uniform variable $U \sim \mathcal{U}(0,1)$, then sets $X = 1$ if $U \leqslant p$, and $X = 0$ otherwise.

```
def RandBer(p, N):
    # p: the probability, N: number of requested samples
    X = np.zeros(N)           # set X = [0,0,...,0] initially
    U = np.random.rand(N)     # generate N uniform numbers in [0,1)
    idx = np.where(U <= p)    # find indices for which U is less than p
    X[idx] = 1                # change the corresponding values in X to 1
    return X
```

Remark 3.1. The `RandDisct` function can be used to generate samples from other discrete distributions. For example, if $X \sim \mathcal{B}in(p,n)$ with pdf

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, \ldots, n,$$

then it is enough to call `X = RandDisct(x,p,N)` for `x` $= [0, 1, \ldots, n]$ and `p` $= [f(0), f(2), \ldots, f(n)]$.
There exist other approaches to generate binomial samples. For example one can generate n iid random variables $X_1, \ldots, X_n$ from $\mathcal{B}er(p)$ and set $X = X_1 + \cdots + X_n$.

Remark 3.2. The `numpy.random` module provides a variety of built-in functions for generating random samples from commonly used distributions. For discrete distributions, you can use the function `numpy.random.choice` instead of the `RandDisct` function.

3.2 Acceptance-Rejection method

Usually, the inverse of the cdf F is not available explicitly, and a numerical inversion might be costly and inefficient. This will make the application of the inverse transform method limited. The *acceptance-rejection* method, introduced by Stan Ulam and John von Neumann, is a more general method that can be used instead.

Suppose that we want to sample from a bounded pdf f which is defined on some finite interval $[a,b]$ and is zero outside this interval. Suppose further that we have an efficient method for sampling from another random variable with pdf g. We follow [**Rubinnstein-Kroese:2017**] and for simplicity we first assume that $g(x) = 1$ on $[a,b]$ and

$$c = \sup\{f(x) : x \in [a,b]\}.$$

The graph of f is clearly under (dominated by) the graph of $cg(x) \equiv c$. See the left panel in Figure 7.

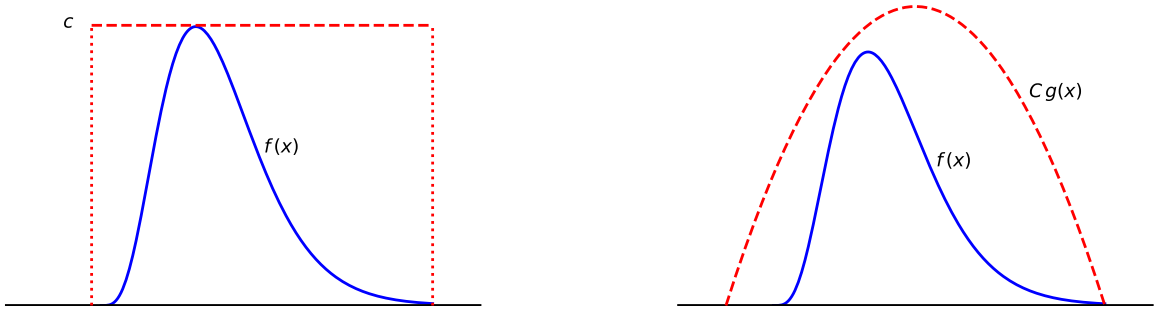


Figure 7: Bounding a pdf $f(x)$ by a function $\phi(x) = Cg(x)$.

Now, we can generate a random variable $X \sim f$ by using the following algorithm.

Algorithm 2 Acceptance-Rejection Algorithm 1

Require: Distribution f on interval $[a, b]$, Constant c

1. Generate $X \sim \mathcal{U}(a, b)$
2. Generate $Y \sim \mathcal{U}(0, c)$ independent of X
3. If $Y \leqslant f(X)$, accept X. Otherwise return to step 1.

Ensure: Random point X from distribution f

Since X and Y are uniformly distributed, the pair (X, Y) is uniformly distributed on rectangle $[a, b] \times [0, c]$. In this rectangle, points (X, Y) that lie under the graph of f are accepted according to the criterion in step 3 of Algorithm 2, while the others are rejected. As a result, the points that are accepted are uniformly distributed in the region under the graph of f. This means that the distribution of the accepted values X is f. The proof will come soon. See Figure 8 for an illustration.

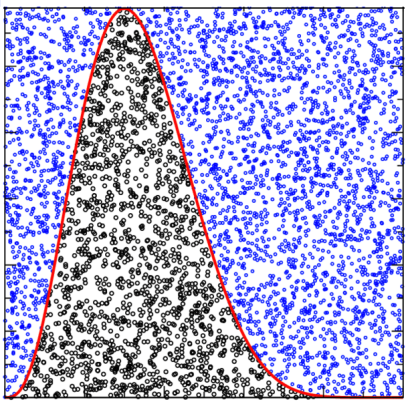


Figure 8: The graph of pdf f (red curve), the bivariate uniformly distributed points on the rectangle (black and blue points together), and accepted (black) and rejected (blue) points. The x-component of accepted points are f-distributed.

Sometimes, as we observe from Figure 8, this algorithm produces many rejected points, which slows down the process. To overcome this inefficiency, we can replace the constant c with a general function $\phi(x) = Cg(x)$, where g is a pdf from which random variables can be easily generated. The pdf g and the constant $C \geqslant 1$ must be chosen carefully so that the graph of f lies under the graph of ϕ while ensuring they are as close as possible to minimize the number of rejected points thereby improve the efficiency of the algorithm. See the right panel of Figure 7. The distribution g is called the *proposal distribution*. The algorithm is given below.

Algorithm 3 Acceptance-Rejection Algorithm 2

Require: Distribution f, Proposal distribution g, Constant C

1. Generate $X \sim g$
2. Generate $Y \sim \mathcal{U}(0, Cg(X))$ independent of X
3. If $Y \leqslant f(X)$, accept X. Otherwise return to step 1.

Ensure: Random point X from distribution f

The random variable generated from this procedure has indeed the desired pdf f. Because if we denote the area under graph $\phi(x) = Cg(x)$ by $\mathcal{A}$ and the area under graph $f(x)$ by $\mathcal{B}$, then the steps 1 and 2 of the above procedure imply that the random variable (X, Y) is uniformly distributed on $\mathcal{A}$. To prove this let $h(x, y)$ be the joint pdf of (X, Y). Then we have

$$h(x, y) = h(y|x)g(x), \quad (x, y) \in \mathcal{A}.$$

Item 2 shows that $h(y|x) = \frac{1}{Cg(x)}$ for $y \in [0, Cg(x)]$. Therefore, $h(x, y) = 1/C$ for $(x, y) \in \mathcal{A}$ which proves that (X, Y) is uniformly distributed on $\mathcal{A}$. This shows that an accepted variable $(\tilde{X}, \tilde{Y})$ is uniformly distributed on $\mathcal{B}$. Since the area of $\mathcal{B}$ is unity, the pdf of $(\tilde{X}, \tilde{Y})$ is 1. The marginal pdf of $Z = \tilde{X}$ on $\mathcal{B}$ is

$$\int_0^{f(x)} 1 \, \mathrm{d}y = f(x),$$

which completes the proof. The efficiency of the algorithm is quantifies by

$$\mathbb{P}((X, Y) \text{ is accepted}) = \frac{\text{area of } \mathcal{B}}{\text{area of } \mathcal{A}} = \frac{1}{C},$$

which means that for a greater efficiency g should be as close as possible to f in order to be able to choose a constant C close to 1.

In item 2 of Algorithm 3 we have $Y \sim \mathcal{U}(0, Cg(x))$ which can be rewritten as $Y = UCg(X)$ where $U \sim \mathcal{U}(0, 1)$. Using this, item 3 can be replaced by $U \leqslant f(X)/(Cg(X))$. Theses result in a new version for the algorithm.

Algorithm 4 Acceptance-Rejection Algorithm 3

Require: Distribution f, Proposal distribution g, Constant C

1. Generate $X \sim g$
2. Generate $U = \mathcal{U}(0, 1)$
3. If $U \leqslant f(X)/(Cg(X))$, accept X. Otherwise return to step 1.

Ensure: Random point X from distribution f

A Python code is given here. The input arguments `f` and `g` are desired and proposal distributions, `C` is the constant factor, and `N` is the number of samples we ask. The input `gGen` is an independent function which draws random samples from g. Finally, `arg_gGen` are input arguments required to execute `gGen`. The output is the vector `Z` containing N samples with pdf f.

```
def RandAcceptReject(f, g, C, N, gGen, *arg_gGen):
    Z = np.zeros(N)
    for k in range(N):
        reject = True
        while reject:
            X = gGen(*arg_gGen)
            U =  np.random.rand()
            if U <= f(X)/(C*g(X)):
                Z[k] = X
                reject = False
    return Z
```

Example 3.3. Consider again the pdf f from Example 3.1. To draw samples from this pdf assume that $g(x) = 1$ and $C = 2$. Since $f(x)/(Cg(x)) = x$, we generate uniform variables $X \sim \mathcal{U}(0,1)$ and $U \sim \mathcal{U}(0,1)$, and accept X if $U \leqslant X$. If not, we repeat the process until receiving an acceptance. Using the following code we generate $N = 500$ and 5000 random points from f and plot the histograms. See Figure 9.

```
import numpy as np
import matplotlib.pyplot as plt
f = lambda x: 2*x
g = lambda x: 1
def gGen(a, b, N):
    U = np.random.uniform(a, b, N)
    return U
arg_gGen = 0, 1, 1
C, N = 2, 500
X = RandAcceptReject(f, g, C, N, gGen, *arg_gGen)
plt.figure(figsize = (5, 3))
plt.hist(X, bins = 30, histtype = 'bar', color = 'red', density = 'true')
x = np.linspace(0,1,200)
plt.plot(x,f(x),linestyle = '-', color = 'blue')
```

```
plt.title('Histogram of $X$ and the pdf $f(x)$')
plt.xlabel('$X$')
plt.ylabel('Frequency %')
```

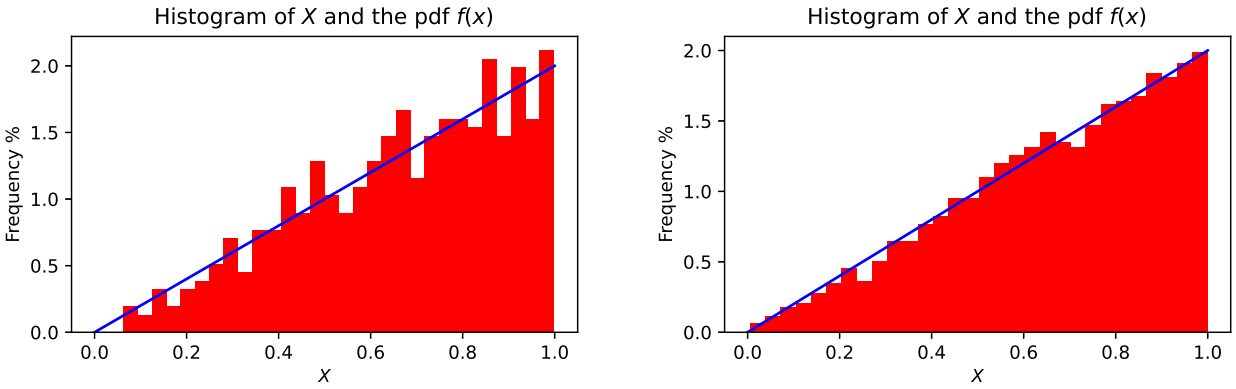


Figure 9: Histograms of generated random variables with pdf f in Example 3.1 using the acceptance-rejection algorithm. $N = 500$ (left), $N = 5000$ (right)

The acceptance-rejection method can also be used to draw samples from the standard normal distribution. The positive portion of the normal standard pdf (with $\mu = 0$ and $\sigma = 1$) can be dominated by a constant C times the pdf of the exponential distribution. We can generate a positive random variable X from the pdf

$$f(x) = \sqrt{\frac{2}{\pi}} \exp\left(-x^2/2\right), \quad x \geqslant 0$$

and then assign it a random sign. The sign can be sampled from the Bernoulli distribution. We assume that $g(x) = \exp(-x)$ which is the pdf of $\mathcal{E}xp(1)$, and $C = \sqrt{2e/\pi}$ to have $f(x) \leqslant Cg(x)$. See Figure 10.

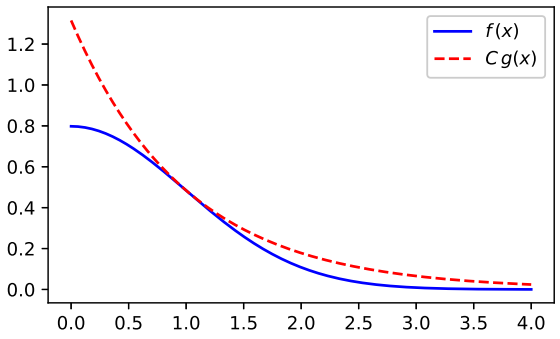


Figure 10: Bounding the positive part of the standard normal distribution $f(x)$ by a constant times the exponential distribution $g(x) = \exp(-x)$.

The acceptance-rejection algorithm begins with generating a random variable $X \sim \mathcal{E}xp(1)$.

The acceptance condition then is

$$U \leqslant \frac{f(X)}{Cg(X)} = \exp\Big(-(X-1)^2/2\Big),$$

or equivalently

$$-\ln U \geqslant \frac{(X-1)^2}{2}.$$

From (3.2) we know that $-\ln U \sim \mathcal{E}xp(1)$. Thus the last inequality can be rewritten as

$$V_1 \geqslant \frac{(V_2-1)^2}{2}$$

where V_1 and V_2 are independent and both of $\mathcal{E}xp(1)$ distribution.

Finally we note that a short section on generating from multivariate distributions (in particular the multivariate normal distribution) is given in Appendix A.9.

4 Monte Carlo method – II

As we observed in section 2, Monte Carlo uses random points to estimate the mean of (complicated) random variables/processes. For continuous random variables it is equivalent to solving certain integrals. If X is a continuous random variable with pdf $f(x)$, and $g(X)$ is some function of X, then $g(X)$ becomes a new random variable. The expectation of $g(X)$ is given by (refer to (A.7) in the Appendix)

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)\,\mathrm{d}x.$$

The Monte Carlo method is an stochastic tool to approximate such integrals numerically, particularly when it is hard to apply the available deterministic methods (e.g. in high-dimensional spaces or on complex domains). The process is called *Monte Carlo integration*. Consider the generic integral

$$\int_a^b g(x)f(x)\,\mathrm{d}x$$

where $f(x)$ is a pdf associated with a random variable X, and $g(x)$ is some function, often referred to as the *performance function*. This integral can be interpreted as the expectation $\mathbb{E}[g(X)]$ of the random variable $g(X)$, where X is distributed according to the pdf $f(x)$.

The expected value can be estimated by drawing random samples from distribution $f(x)$, evaluating $g(x)$ at these random points, and then averaging the results. Specifically, the Monte Carlo procedure follows these steps:

1. Generate N random samples $x_1, x_2, \ldots, x_N$ from pdf $f(x)$.
2. For each sample x_k, compute the corresponding value $g(x_k)$.
3. Approximate the integral by computing the average of these values:

$$\frac{1}{N}\sum_{k=1}^{N} g(x_k) =: \overline{g}_N.$$

As N increases, this approximation converges to the true value of the integral, thanks to the

law of large numbers. The convergence proof will come soon.

Example 4.1. Consider the following 1D integral

$$I = \int_0^1 g(x)\,dx.$$

This integral can be interpreted as the expectation of $g(X)$, where X is a uniformly distributed random variable on the interval $[0, 1]$. In other words

$$I = \mathbb{E}[g(X)] = \int_0^1 g(x) \cdot 1\,dx, \quad X \sim \mathcal{U}(0, 1).$$

The Monte Carlo method generates N random points $x_1, x_2, \ldots, x_N$ from distribution $\mathcal{U}(0, 1)$, and computes

$$I \approx \frac{1}{N}\left(g(x_1) + g(x_2) + \cdots + g(x_N)\right)$$

A code snippet is given below for $g = \sin x$.

```
import numpy as np
def g_fun(x):
    return np.sin(x)                # the integrand g(x) = sin(x)
int_exact = 1-np.cos(1)             # the exact value for comparison
int_mc = np.zeros(6)
for k in range(6):
    N = 10**k                       # number of points from 1, 10,..., 10^5
    X = np.random.uniform(0, 1, N) # generate uniform random points in [0,1]
    g = g_fun(X)                    # evaluate the integrand on X
    int_mc[k] = np.mean(g)          # take mean
print('int_mc = ',
      np.round(np.abs(int_exact-int_mc),5)) # errors, rounded to 5 decimals
```

An execution gives

```
int_mc =  [0.03652 0.08578 0.02132 0.00362 0.00203 0.00063]
```

A new execution will result in a new (different) error vector as the integral points are generated randomly.

```
int_mc =  [0.03011 0.00532 0.00749 0.01049 0.00279 0.00023]
```

In any case, the accuracy improves as the number of random samples N increases. However, the convergence speed is slow.

In general to estimate the integral of a function g on finite interval $[a, b]$ the Monte Carlo integration is applied as below:

$$\int_a^b g(x)dx = (b-a)\int_a^b g(x)\frac{1}{b-a}dx \approx \underbrace{(b-a)}_{width}\underbrace{\frac{1}{N}\sum_{k=1}^{N} g(x_k)}_{mean\ \ hight}, \quad x_k \in \mathcal{U}(a,b)$$

See also Figure 11 for an illustration.

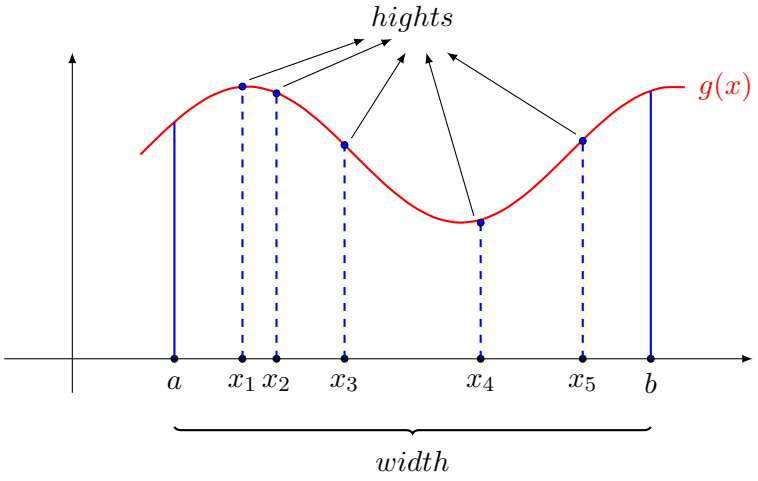


Figure 11: Schematic of 1D Monte Carlo integration

Example 4.2. To estimate the value of integral

$$I = \int_{-\infty}^{\infty} (x^4 - x + 1)\mathrm{e}^{-x^2/2}\mathrm{d}x$$

using Monte Carlo method, we can write

$$I = \sqrt{2\pi}\int_{-\infty}^{\infty} \underbrace{(x^4 - x + 1)}_{g(x)}\underbrace{\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-x^2/2}}_{f(x)}\,dx$$

where $f(x)$ is the pdf of the standard normal distribution on $(-\infty, \infty)$. The estimate then is

$$I \approx \sqrt{2\pi} \times \frac{1}{N}\sum_{k=1}^{N}(x_k^4 - x_k + 1)$$

where x_k are generated from $\mathcal{N}(0, 1)$.

4.1 Convergence of Monte Carlo integration

Before providing an analysis for convergence of the Monte Carlo method, we first compare its convergence rate with a deterministic method for computing a one-dimensional integral. As a simple deterministic integration quadrature consider the mid-point (MP) rule:

$$\begin{aligned}\int_0^1 g(x)\mathrm{d}x &= h\left[g(x_1^*) + g(x_2^*) + \cdots + g(x_N^*)\right] + \mathcal{O}(h^2)\\ &= \frac{1}{N}\left[g(x_1^*) + g(x_2^*) + \cdots + g(x_N^*)\right] + \mathcal{O}(N^{-2})\end{aligned}$$

where $h = 1/N$ and N is the number of integration points in the interval $[0, 1]$, and $x_k^* = (x_{k-1} + x_k)/2$ are mid points. Compared to the Monte Carlo method, the integration points x_k^* in the mid-point rule are a set of *equidistance* points. For a sufficiently smooth function, say $g \in C^2[a, b]$, the order of convergence of this method is 2 — if h is halved (equivalently if N is doubled) then the error is quartered. The error plot for both methods are given in Figure 12. For Monte Carlo, the results of ten simulations are depicted and the approximate average rate is computed, which is close to 0.5.

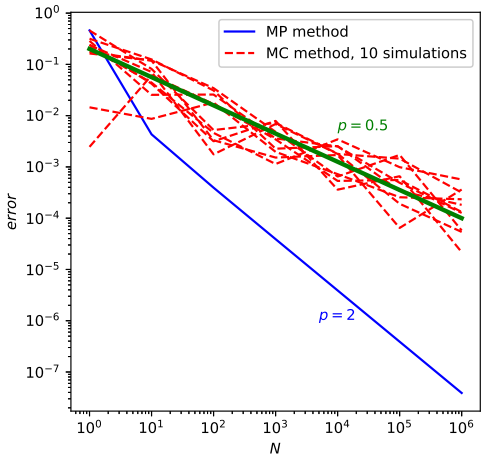


Figure 12: Convergence orders of Monte Carlo (MC) and mid-point (MP) methods for integration.

The convergence of the mid-point rule is of order h^2, or equivalently N^{-2}. When we extend this rule to a two-dimensional integral on a rectangular domain, the convergence remains at order h^2, which translates to N^{-1} since, in two dimensions, h is proportional to $N^{-1/2}$. In general, for a d-variate integral, the convergence of the mid-point method is of order $N^{-2/d}$ in the *maximum norm*. This reduction in the convergence order with increasing dimensions also occurs in other deterministic methods, such as the trapezoidal method and Simpson's method. However, as we will see, the convergence of the Monte Carlo method is *almost surely* of order $N^{-1/2}$, *independent of the dimension*. This indicates that while deterministic methods, if practically applicable, are preferred for low-dimensional integrals, the Monte Carlo method is often a better choice for high-dimensional integrals.

In order to determine how accurate the Monte Carlo solution is, we assume that X is a random variable with pdf f and $X_1, X_2, \ldots, X_N$ is a sample (a set of independent and identically distributed (iid) random variables) from X. For a function g, assume that $g(X)$, as another random variable, has the mean μ and variance σ^2, i.e.,

$$\mu = \mathbb{E}_f[g(X)], \quad \sigma^2 = \mathrm{Var}_f[g(X)].$$

Since, X_k are iid, all $g(X_k)$ have the same mean μ and variance σ^2. The new random variable

$$Y = \frac{1}{N} \sum_{k=1}^{N} g(X_k), \quad X_k \sim f \tag{4.1}$$

is an *unbiased* estimator for $\mu = \mathbb{E}[g(X)]$ in the sense that

$$\mathbb{E}(Y) = \frac{1}{N}\sum_{k=1}^{N}\mathbb{E}[g(X_k)] = \frac{N\mu}{N} = \mu.$$

What can we say about the variance of Y? Using the *central limit theorem* (see Appendix A.10), for sufficiently large values of N we have

$$Y \sim \mathcal{N}(\mu, \sigma^2/N) \quad \text{or} \quad \sqrt{N}\frac{Y-\mu}{\sigma} \sim \mathcal{N}(0,1).$$

This shows that the variance of Y is σ^2/N, proving that Y approaches μ in probability with convergence rate $\mathcal{O}(1/\sqrt{N})$. More precisely, assume that Φ denotes the standard normal cdf and z_γ denotes the γ-quantile of $\mathcal{N}(0,1)$, i.e., $\Phi(z_\gamma) = \gamma$. This means that if $Z \sim \mathcal{N}(0,1)$ then

$$\mathbb{P}(-z_{1-\alpha/2} \leqslant Z \leqslant z_{1-\alpha/2}) = 1-\alpha.$$

Thus, we can write

$$\mathbb{P}\left(-z_{1-\alpha/2} \leqslant \frac{\sqrt{N}(Y-\mu)}{\sigma} \leqslant z_{1-\alpha/2}\right) = 1-\alpha.$$

or equivalently

$$\mathbb{P}\left(\mu - z_{1-\alpha/2}\frac{\sigma}{\sqrt{N}} \leqslant Y \leqslant \mu + z_{1-\alpha/2}\frac{\sigma}{\sqrt{N}}\right) = 1-\alpha.$$

In other words, with probability $(1-\alpha)100\%$ the *confidence interval* for Y is

$$\left[\mu - z_{1-\alpha/2}\frac{\sigma}{\sqrt{N}}\ ,\ \mu + z_{1-\alpha/2}\frac{\sigma}{\sqrt{N}}\right].$$

See Figure 13. For example, if $\alpha = 0.05$ then $z_{1-\alpha/2} = z_{0.975} \doteq 1.96$, which means that with

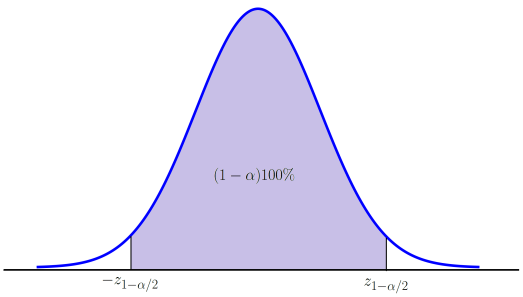


Figure 13: Confidence interval in the standard normal distribution

95% probability Y falls in interval

$$\left[\mu - 1.96\frac{\sigma}{\sqrt{N}}\ ,\ \mu + 1.96\frac{\sigma}{\sqrt{N}}\right].$$

This probability will increase to 99% if we replace the factor 1.96 in the confidence interval by $z_{0.995} \doteq 2.576$, and to 0.999% by $z_{0.9995} \doteq 3.29$. In general, the accuracy of the estimator Y is determined by its standard deviation, i.e., $\sigma/\sqrt{N}$.

We again emphasis that μ and σ^2 are the mean and the variance of random variable $g(X)$. Our aim was to estimate μ but for error estimation the value of σ is also required. Usually, σ^2

is unknown, but can be estimated with the *sample variance*

$$s_N^2 = \frac{1}{N-1}\sum_{k=1}^{N}(g(x_k) - \overline{g}_N)^2, \quad \overline{g}_N = \frac{1}{N}\sum_{k=1}^{N} g(x_k)$$

where x_k are generated from pdf f. The sample variance s_N^2 tends to σ^2 by the law of large numbers. Consequently, for large values of N, the *approximate* confidence interval for Y is

$$\left[\mu - z_{1-\alpha/2}\frac{s_N}{\sqrt{N}}\ ,\ \mu + z_{1-\alpha/2}\frac{s_N}{\sqrt{N}}\right].$$

We must be aware that the confidence interval can be trusted as far as s_N^2 is a proper estimate for the variance σ^2.

Example 4.3. To estimate the cdf of the standard normal distribution, i.e.,

$$\Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2} \mathrm{d}x \tag{4.2}$$

using the Monte Carlo method, we generate N normal variable $X_1, \ldots, X_N \sim \mathcal{N}(0,1)$ and set

$$\overline{\Phi}(t) = \frac{1}{N}\sum_{k=1}^{N} \mathbb{I}_{X_k \leqslant t}, \tag{4.3}$$

with (exact) variance $\Phi(t)[1-\Phi(t)]/N =: \sigma^2/N$, since the variables $\mathbb{I}_{X_k \leqslant t}$ are independent Bernoulli variables with success probability $\Phi(t)$. Here, $\mathbb{I}_A = 1$ or 0 when A is true or false, respectively. The confidence interval tells that with $(1-\alpha)100\%$ probability the error is at most $z_{1-\alpha/2}\sigma/\sqrt{N}$ (or approximately $z_{1-\alpha/2}s_N/\sqrt{N}$). For example, for $t = 0$ we have $\Phi(t) = 0.5$ and $\sigma^2 = \Phi(t)[1-\Phi(t)] = 1/4$. For this special case, to achieve a precision of three decimals with probability 95%, we set $\alpha = 0.05$, giving $z_{1-\alpha/2} \doteq 1.96$, and choose N such that

$$\frac{z_{1-\alpha/2}\sigma}{\sqrt{N}} \doteq \frac{1.96}{2\sqrt{N}} \leqslant \frac{1}{2}10^{-3}.$$

which gives $N > 3.85 \cdot 10^6$. Experimental results for different values of N and t are obtained by executing the following code.

```
import numpy as np
from scipy.stats import norm
t, K = 0, 7
Phi, PhiBar = norm.cdf(t), np.zeros(K)
for k in range(K):
    N = 10**(k+1)
    X = np.random.normal(0,1,N)
    PhiBar[k] = 1/N*len(X[(X < t)])
Error = abs(PhiBar-Phi)/Phi
print('Phi(',t,') = ',Phi,'\n','PhiBar = ',PhiBar,'\n','Error = ',Error)
```

Here, an output of this code for $t = 0$ is given. We again note that $\Phi(0) = 0.5$. The last estimation 0.5002081 with relative error 0.0004162 corresponds to $N = 10^7$ standard normal

samples, which confirms our error estimation above.

```
Phi(0) =  0.5
PhiBar =  [0.7  0.46  0.516  0.4983  0.50105  0.500231  0.5002081]
Error  =  [0.4  0.08  0.032  0.0034  0.00210  0.000462  0.0004162]
```

However, if we execute the code for $t = -4.5$, i.e., to estimate $\Phi(-4.5) \doteq 3.39767 \times 10^{-6}$, the following results will be obtained.

```
Phi(-4.5) =  3.3976731247300535e-06
PhiBar =  [0.0e+00 0.0e+00 0.0e+00 0.0e+00 0.0e+00 4.0e-06 3.6e-06]
Error  =  [1.      1.      1.      1.      1.      1.8e-01 6.0e-02]
```

In this experiment, we observe that for values of N up to 10^5, the estimated values remain at 0, resulting in errors of 100%. This outcome indicates that none of the generated random points fall before $x = -4.5$. For larger values of N, such as 10^6 and 10^7, the estimations are nonzero but still quite poor. The issue lies in the fact that we are attempting to estimate the probability of a very rare event. To have enough samples from f in such sub-domains ($x \leq -4.5$, the left tail of normal distribution), a huge number of samples is needed in the whole domain. This is a drawback of the basic Monte Carlo method. However, by applying a modification we can significantly improve the accuracy with fewer samples. This is discussed in the next subsection.

Exercise 4.1. The irrational number π is the volume of the unit ball in $\mathbb{R}^2$. We can compute the area of the first quadrant sector and multiply it by 4:

$$\pi = 4 \int_0^1 \sqrt{1 - x^2} \mathrm{d}x.$$

Use Monte Carlo to estimate π by approximating the above integral with different number of samples. Compare the errors with the Monte Carlo error bound. Plot the errors in log-log scale for 10 executions.

Exercise 4.2. Estimate the integral

$$I = \int_0^\infty \mathrm{e}^{-0.5x}(x^2 - x)\mathrm{d}x$$

using the Monte Carlo integration with different values of N. Identify the right distribution to sample from for this estimation. You can compute the exact value of this integral using the integration by parts. Compare the exact value and the Monte Carlo solutions. Are your results confirmed by the error bound of the Monte Carlo method?

4.2 Importance sampling

The importance sampling approach is based on the principle that the expectation of $g(X)$ with respect to density f can be written in the alternative form

$$\mathbb{E}_f[g(X)] = \int g(x)f(x)\mathrm{d}x = \int g(x)\frac{f(x)}{\ell(x)}\ell(x)\mathrm{d}x = \mathbb{E}_\ell\left[g(X)\frac{f(X)}{\ell(X)}\right] \tag{4.4}$$

where $\ell(x)$ is another density function, called the *importance sampling function* or *envelope*. Equation (4.4) suggests to estimate $\mathbb{E}_f[g(X)]$ by drawing iid random variables $X_1, \ldots, X_N$ from ℓ (not f!) and use the estimator

$$Y_{\mathrm{IS}} = \frac{1}{N}\sum_{k=1}^{N} g(X_k)\frac{f(X_k)}{\ell(X_k)}, \quad X_k \sim \ell \tag{4.5}$$

instead of basic estimator (4.1). Here, the subscript 'IS' stands for 'Importance Sampling'. For this strategy to work, it must be easy to sample from ℓ and to evaluate f, even when it is not easy to sample from f. Indeed, (4.5) does converge to $\mathbb{E}_f[g(X)]$ for the same reason the basic Monte Carlo estimator converges, whatever the choice of the distribution ℓ is, as long as $\mathrm{supp}(\ell) \subset \mathrm{supp}(g \times f)$. This assumption on the support of ℓ is important because a smaller support truncates the integral (4.4) and produces a biased result. The ratio of densities is denoted by

$$w(x) = \frac{f(x)}{\ell(x)},$$

and is called the *likelihood* ratio. This ratio needs only to be known up to a constant; let say $w(x) = c\tilde{w}(x)$. This is useful, for example, when the distribution f or ℓ is known only up to a constant. Since $\mathbb{E}_\ell[w(X)] = 1$ we can write

$$\mathbb{E}_f[g(X)] = \mathbb{E}_\ell\left[g(X)w(X)\right] = \frac{\mathbb{E}_\ell\left[g(X)w(X)\right]}{\mathbb{E}_\ell\left[w(X)\right]},$$

which motivates the *weighted sample estimator*

$$\frac{\frac{1}{N}\sum_{k=1}^{N} g(X_k)w(X_k)}{\frac{1}{N}\sum_{k=1}^{N} w(X_k)} = \frac{\sum_{k=1}^{N} w_k g(X_k)}{\sum_{k=1}^{N} w_k} =: Y_{\mathrm{IS}}^{w} \tag{4.6}$$

where $w_k = \tilde{w}(X_k)$. Note that the estimators Y_{IS} and Y_{IS}^{w} are mathematically the same but Y_{IS}^{w} can also be used in situations where either f or ℓ is missing a normalizing constant. Both number N and constant c are cancelled from the numerator and denominator in (4.6).

Example 4.4. Coming back to Example 4.3, consider again the estimation of $\Phi(t)$, the cdf of the standard normal distribution. As we observed, the basic Monte Carlo method fails to produce accurate estimation for $\Phi(-4.5)$, at least for values of N smaller than 10^6. Here, we apply an importance sampling estimator. Let f be the pdf of the standard normal distribution. We consider a distribution ℓ with a support restricted to $(-\infty, -4.5]$ to remove the unnecessary variation of the basic Monte Carlo estimator due to simulating zero values for $x > -4.5$. A good choice is to take $\ell(x) = \tilde{\ell}(-x - 4.5)$ for $x \in (-\infty, -4.5]$, where $\tilde{\ell}$ is

the pdf of the exponential distribution $\mathcal{E}xp(1)$, i.e.

$$\ell(x) = \exp(x + 4.5), \quad x \leqslant -4.5$$

See Figure 14.

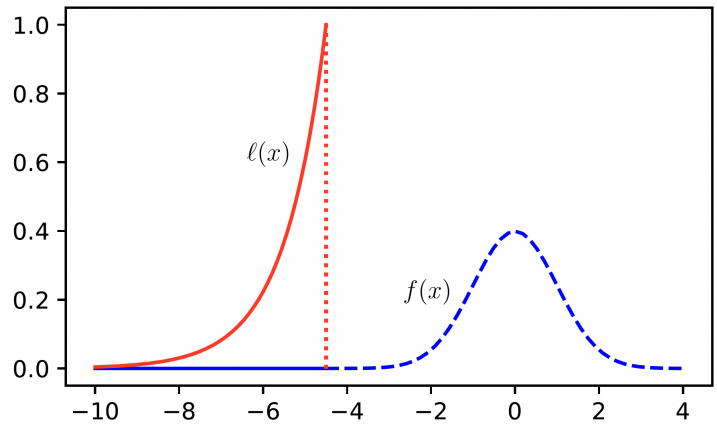


Figure 14: The primary pdf f and the importance sampling pdf ℓ.

The support of ℓ is the same as the integral domain in (4.2) for $t = -4.5$. If we generate random variables $X_1, \ldots, X_N$ from ℓ then $g(X_k) = \mathbb{I}_{X_k \leqslant -4.5} = 1$ for all $k = 1, \ldots, N$ and the estimation (4.5) becomes

$$\overline{\Phi}_{\mathrm{IS}}(t) = \frac{1}{N} \sum_{k=1}^{N} \frac{f(X_k)}{\ell(X_k)}, \tag{4.7}$$

for $t = -4.5$. The code and the outputs are given below.

```
t = -4.5
import numpy as np
from scipy.stats import norm, expon   # normal and exponential dist.
K = 7
Phi, PhiBar = norm.cdf(t), np.zeros(K)
for k in range(K):
    N = 10**(k+1)
    X = -RandExp(1,N) + t
    gX = 1
    fX = norm.pdf(X, loc = 0, scale = 1)
    ellX = expon.pdf(-(X-t), scale = 1)
    PhiBar[k] = 1/N*np.sum(gX*fX/ellX)
Error = abs(PhiBar-Phi)/Phi
print(' Phi(',t,')=',Phi,'\n','PhiBar=',PhiBar,'\n','Error =',Error)
```

Outputs:

```
Phi(-4.5)=3.398e-06
PhiBar=[2.068e-6 4.084e-6 3.332e-6 3.417e-6 3.388e-6 3.396e-6 3.397e-6]
Error =[3.912e-1 2.019e-1 1.946e-2 5.720e-3 2.958e-3 4.943e-4 1.252e-4]
```

We observe more accurate estimations compared with the results obtained at the end of Example 4.3 using the basic Monte Carlo method.

To analyze the results of Examples 4.3 and 4.4 we can look at the variances of estimators (4.1) for basic Monte Carlo and (4.5) for Monte Carlo with importance sampling. Since X_k are assumed to be independent random variables from either f or ℓ, the variance of Y or Y_{IS} is the sum of the variances of the individual terms divided by N. The variances of individual variables in (4.1) and (4.5) are

$$\text{Var}_f[g(X)] = \mathbb{E}_f(g^2(X)) - (\mathbb{E}_f[g(X)])^2 = \int g^2(x)f(x)\mathrm{d}x - \left(\int g(x)f(x)\mathrm{d}x\right)^2,$$
$$\text{Var}_\ell\left[g(X)\frac{f(X)}{\ell(X)}\right] = \int g^2(x)\frac{f^2(x)}{\ell(x)}\mathrm{d}x - \left(\int g(x)f(x)\mathrm{d}x\right)^2,$$

respectively. The second terms (squares of expectations) are the same for both variances. However, depending on the importance sampling function ℓ, the first term in the second variance can be smaller than the first term in the first variance. To have this, we assume that $f^2(x)/\ell(x) \leqslant f(x)$ or equivalently, $f(x) \leqslant \ell(x)$ for all x in the integration domain. This means that ℓ should not have a "lighter tail" than f. See the graphs of f and ℓ in Figure 14.

Exercise 4.3. Coming back to Examples 4.3 and 4.4, show that for $t = -4.5$ we have

$$\text{Var}_f(\overline{\Phi}) = \Phi(-4.5)[1 - \Phi(-4.5)]/N \doteq \frac{3.3977}{N} \times 10^{-6},$$
$$\text{Var}_\ell\left(\overline{\Phi}_{\text{IS}}\right) = \left[\exp(-4)/(2N\sqrt{\pi})\Phi(-4\sqrt{2})\right] - [\Phi(-4.5)]^2 \doteq \frac{3.8373}{N} \times 10^{-11}.$$

What do you conclude from this comparison?

Exercise 4.4. Execute the Python code of Example 4.4 with $t = 0$ instead of $t = -4.5$. Compare the outputs with those of Example 4.3. Report your observation and try to give an analysis.

To find a proper importance sampling function ℓ, one may try to minimize

$$\text{Var}_\ell\left[g(X)\frac{f(X)}{\ell(X)}\right]$$

over all possible functions ℓ. We can show that the solution of this minimization problem is

$$\ell^*(x) = \frac{|g(x)|f(x)}{\int |g(x)|f(x)\mathrm{d}x},$$

and in particular case $g(x) \geqslant 0$,

$$\ell^*(x) = \frac{g(x)f(x)}{\mathbb{E}_f[g(X)]}.$$

This sampling function results in a zero variance for its corresponding importance sampling estimator. However, the optimal density ℓ^* is not practical because deriving ℓ^* requires knowing $\mathbb{E}_f[g(X)]$, which is the quantity we are trying to estimate it! Additionally, in some cases, the explicit form of the performance function $g(x)$ may not be known in advance. In practice, we try to approximate ℓ^* using sample values $g(X_1), \ldots, g(X_N)$. We do not pursue this further and refer you to more advanced texts in Monte Carlo simulation.

5 Stochastic processes

A stochastic process is a family of random variables evolving in time. For example, each stochastic solution of the radioactive decay problem given on the right-hand side of Figure 1 is a stochastic process. When it comes to compare with the deterministic case, the solution of a deterministic model (e.g. a differential equation) is a *function* while the solution of its stochastic counterpart is a *stochastic process*. In practice, we can have a set of stochastic processes as solutions, and a possibility is to use the Monte Carlo method to average them and provide an estimate of the expected behavior of the system.

Definition 5.1. A family of random variables $\{X_t : t \in \mathcal{T}\}$, all defined on the same probability space, is called a *stochastic process* or random process. If $\mathcal{T} = \{0, 1, 2, \ldots\} =: \mathbb{N}_0$ the sequence $X_0, X_1, \ldots$ is called a stochastic process with *discrete time parameter*, and if $\mathcal{T} = [0, \infty)$ the family is called a stochastic process with *continuous time parameter*. The first random variable X_0 is called the *initial state* of the process; and the random variable X_t for a $t \in \mathcal{T}$ is called the *state* of the process at time t.

We note that the index family $\mathcal{T}$ determines the discrete and continuous nature of the stochastic process. Independent of this, random variables X_t (states) may have either discrete or continuous probability density functions.

Example 5.1. Suppose that a certain business office has five telephone lines, each one of which might be in use at any given time. At discrete times $t = 0, 1, 2, \ldots$ minutes the telephone lines are observed and the number of lines that are being used at each time is noted. Let X_0 denote the number of lines that are being used at the first time, let X_1 denote the number of lines that are being used at the second time, 1 minutes later; and in general

$$X_t = \text{ the number of lines that are being used when they are observed time } t.$$

Then $X_0, X_1, X_2, \ldots$, is stochastic process with discrete time parameter. The state X_t of the process at any discrete time t is the number of lines being used at that time. Therefore,

each state must be an integer between 0 and 5.

Example 5.2. A coin purse contains 5 quarters (each worth 25¢), 5 dimes (each 10¢) and 5 nickels (each 5¢). Assume that we draw coins one by one and set on a table. Let

$$X_t = \text{ total value of coins set on the table after } t \text{ draws.}$$

We see that $\{X_t,\ t = 0, 1, 2, \ldots\}$ is a stochastic process with discrete time variable $t = 0, 1, 2, \ldots$ with initial state $X_0 = 0$.
Assume that in the first 6 draws, 3 nickels and 1 quarter and 2 dimes are drawn with

$$\underset{\text{quarter}}{X_1 = 25}, \quad \underset{\text{nickel}}{X_2 = 30}, \quad \underset{\text{nickel}}{X_3 = 35}, \quad \underset{\text{dime}}{X_4 = 45}, \quad \underset{\text{nickel}}{X_5 = 50}, \quad \underset{\text{dime}}{X_6 = 60}.$$

What is the probability of $X_7 = 65$ given the above information? We are left with 4 quarters, 3 dimes and 2 nickels. We simply see that $\mathbb{P}(X_7 = 65 \,|\, \text{all information above}) = 2/9$ and $\mathbb{P}(X_7 = 80 \,|\, \text{all information above}) = 0$. This means that we can predict the future of the process (i.e. the state at time $t + 1$) based on available information at all previous times $t = 0, 1, \ldots, t$.

In a stochastic process with a discrete time parameter, the state of the process varies in a random manner from time to time. To describe a complete probability model for a particular process, it is necessary to specify the distribution for the initial state X_0 and also to specify for each $t = 0, 1, \ldots$ the conditional distribution of the subsequent state X_{t+1} given $X_1, \ldots, X_t$. These conditional distributions are equivalent to the collection of conditional probabilities of the form

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t).$$

5.1 Markov processes

Markov processes are stochastic processes whose futures are conditionally independent of their pasts given their present values. On the other words, a stochastic process is Markov, if one can make predictions for the future of the process based solely on it's present state. One can say that a Markov process is *memoryless*.

Example 5.3. The stochastic process X_t in Example 5.2 is not a Markov process because any prediction about X_7 would require all information of previous states $X_6, \ldots, X_1$. Now we define a new process

$$Y_t = (q_t, d_t, n_t),$$

where q_t, d_t, and n_t are the counts of quarters, dimes, and nickels, respectively, on the table at time t. For example, $Y_0 = (0, 0, 0)$, $Y_1 = (1, 0, 0)$, $Y_2 = (1, 0, 1)$, ..., $Y_6 = (1, 2, 3)$. The process Y_t is Markov because the probability distribution of Y_{t+1} depends only on the current state Y_t and not the earlier states.

A Markov process with a discrete index set, i.e. $\mathcal{T} = \mathbb{N}_0$ is called a *Markov chain*. The states of a Markov chain can be either discrete (countable) or continuous. For a Markov chain $X = \{X_t : t \in \mathbb{N}_0\}$ with a discrete state space $\mathcal{S}$ we have

$$\mathbb{P}(X_{t+1} = x_{t+1}|X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t) = \mathbb{P}(X_{t+1} = x_{t+1}|X_t = x_t), \tag{5.1}$$

for all $x_0, \ldots, x_{t+1} \in \mathcal{S}$ and $t \in \mathbb{N}_0$. This property says that the conditional distributions of X_{t+1} given $X_1, \ldots, X_t$ depend only on X_t and not on the earlier states $X_1, \ldots, X_{t-1}$.

If the state space $\mathcal{S}$ is finite then the Markov chain is called *finite*. Using the conditional probability rule (A.3), we have

$$\mathbb{P}(X_0 = x_0, X_1 = x_1) = \mathbb{P}(X_0 = x_0)\mathbb{P}(X_1 = x_1|X_0 = x_0). \tag{5.2}$$

For Markov chains, by applying the product rule (A.4) and the Markov chain property (5.1), we can prove the following theorem.

Theorem 5.2. For a finite Markov chain the joint pdf for the first t states is

$$\begin{aligned}&\mathbb{P}(X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t) = \\ &\quad \mathbb{P}(X_0 = x_0)\mathbb{P}(X_1 = x_1|X_0 = x_0)\mathbb{P}(X_2 = x_2|X_1 = x_1)\cdots\mathbb{P}(X_t = x_t|X_{t-1} = x_{t-1}).\end{aligned} \tag{5.3}$$

The proof is straightforward and is left as an exercise. Theorem 5.2 shows that the joint distribution of a finite Markov chain X can be characterized by the distribution of the initial state X_0, i.e., $\mathbb{P}(X_0 = x_0)$, and the *one-step transition probabilities*

$$\mathbb{P}(X_{t+1} = x_{t+1}|X_t = x_t), \quad x_t, x_{t+1} \in \mathcal{S}.$$

In a finite Markov chain we assume that $|\mathcal{S}| = m$. It will also be convenient to name the m states using the integers $1, 2, \ldots, m$ or $0, 1, 2, \ldots, m-1$. Then for each t and j, $X_t = j$ will mean that the chain is in state j at time t. For example, if the states in Example 5.1 are the numbers of phone lines in use at given times, we have $\mathcal{S} = \{0, 1, 2, 3, 4, 5\}$ and $m = 6$.

Definition 5.3. If the transition probabilities $\mathbb{P}(X_{t+1} = j|X_t = i)$ for $i, j \in \mathcal{S}$ are independent of the time then the chain is called *time-homogeneous*.

For a time-homogeneous Markov chain there exist probabilities p_{ij}, independent of t, such that

$$p_{ij} = \mathbb{P}(X_{t+1} = j|X_t = i), \quad i, j \in \mathcal{S}, \quad \forall t \in \mathbb{N}_0. \tag{5.4}$$

We can put this probabilities into a matrix P as

$$P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

which is called the *transition matrix*. If $|\mathcal{S}| = m$ then P is an $m \times m$ matrix. We note that all elements of P are nonnegative and each row sums up to unity. i.e.,

$$\sum_{j=0}^{\infty} p_{ij} = 1,$$

because in the language of multivariate distributions the transition probabilities p_{ij} are indeed conditional density functions of X_{t+1} given X_t which can also be denoted by $g(j|i) = p_{ij}$ for all t and i, j.

Example 5.4. Consider the daily morning weather in a city. Assume there can be three different states: (1) sunny, (2) cloudy, or (3) rainy. Observations of the weather forecasting office show that a sunny day is never followed by another sunny day. Rainy or cloudy weather is equally probable after a sunny day. A rainy or cloudy day is followed by 50% probability by another day with the same weather. If, on the other hand, the weather is changing from cloudy or rainy weather, the following day will be sunny only in half of the cases. Based on this observation the transition matrix is

$$\begin{array}{r} \\ \text{sunny} \\ \text{cloudy} \\ \text{rainy} \end{array} \begin{array}{c} \begin{array}{ccc} \text{sunny} & \text{cloudy} & \text{rainy} \end{array} \\ \begin{bmatrix} 0.00 & 0.50 & 0.50 \\ 0.25 & 0.50 & 0.25 \\ 0.25 & 0.25 & 0.50 \end{bmatrix} \end{array} =: P$$

The transition matrix is independent of time, which means that the transition rules remain unchanged during a period of time; for example during a month. Sometimes it is easier to have a *transition graph* instead of the transition matrix. The transition graph of the above example is shown in Figure 15.

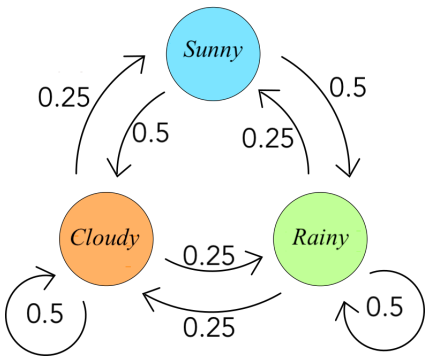


Figure 15: Transition graph of the daily weather example.

Looking at the graph, we observe that if, for example, today is cloudy then tomorrow is sunny with 25% probability, i.e., $\mathbb{P}(X_{t+1} = \textit{sunny} \,|\, X_t = \textit{cloudy}) = 0.25$. The process is Markov. Why?

A vector consisting of nonnegative numbers that sum to 1 is called a probability vector. The *initial distribution* of the chain is also called the *initial probability vector*.

Remark 5.1. Following statistics texts, in this section a vector in $\mathbb{R}^n$ is considered as a $(1 \times n)$ array (a row vector). This violates our notation in previous numerical linear algebra lectures where by a vector we meant a column vector.

For a chain with m possible states $\mathcal{S} = \{1, 2, \ldots, m\}$, the initial probability vector is

$$\boldsymbol{\pi}^{(0)} = \Big[\mathbb{P}(X_0 = 1),\, \mathbb{P}(X_0 = 2), \ldots, \mathbb{P}(X_0 = m)\Big],$$

and the distribution (probability vector) of X at time t is denoted by

$$\boldsymbol{\pi}^{(t)} = \Big[\mathbb{P}(X_t = 1),\, \mathbb{P}(X_t = 2), \ldots, \mathbb{P}(X_t = m)\Big].$$

At $t = 1$, the distribution is identified according to (5.2):

$$\begin{aligned}
\pi_j^{(1)} = \mathbb{P}(X_1 = j) &= \sum_{i=1}^{m} \mathbb{P}(X_1 = j, X_0 = i) \\
&= \sum_{i=1}^{m} \mathbb{P}(X_1 = j | X_0 = i)\mathbb{P}(X_0 = i) \\
&= \sum_{i=1}^{m} p_{ij}\pi_i^{(0)}
\end{aligned}$$

for $j = 1, 2, \ldots, m$. This can be written in a matrix-vector form as

$$\boldsymbol{\pi}^{(1)} = \boldsymbol{\pi}^{(0)} P.$$

In general, we can prove by induction that

$$\boldsymbol{\pi}^{(t)} = \boldsymbol{\pi}^{(t-1)} P = \boldsymbol{\pi}^{(t-2)} PP = \cdots = \boldsymbol{\pi}^{(0)} P^t.$$

Note that, if $\boldsymbol{\pi}^{(t)}$ is a probability vector so is $\boldsymbol{\pi}^{(t+1)}$, because $0 \leqslant p_{ij} \leqslant 1$ and rows of P sum up to 1. We can also show that the t-step transition probabilities are

$$\mathbb{P}(X_t = j | X_0 = i) = p_{ij}^{(t)}, \quad i, j, = 1, 2, \ldots, m$$

where $p_{ij}^{(t)}$ are entries of matrix P^t. The proof for the 2-step transition is as follows

$$\begin{aligned}
\mathbb{P}(X_2 = j | X_0 = i) &= \sum_{k=1}^{m} \mathbb{P}(X_1 = k, X_2 = j | X_0 = i) \\
&= \sum_{k=1}^{m} \mathbb{P}(X_1 = k | X_0 = i)\mathbb{P}(X_2 = j | X_1 = k, X_0 = i) \\
&= \sum_{k=1}^{m} \mathbb{P}(X_1 = k | X_0 = i)\mathbb{P}(X_2 = j | X_1 = k) \qquad \text{(Markov chain property)} \\
&= \sum_{k=1}^{m} p_{ik}p_{kj} = p_{ij}^{(2)}.
\end{aligned}$$

The general case can be proved similarly.

Example 5.5. Consider the transition matrix in Example 5.4. If the weather is cloudy in the day 0 then $\boldsymbol{\pi}^{(0)} = [0, 1, 0]$, and the probability vector $\boldsymbol{\pi}^{(1)}$ is computed by

$$\boldsymbol{\pi}^{(1)} = \boldsymbol{\pi}^{(0)} P = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0.50 & 0.50 \\ 0.25 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \end{bmatrix}.$$

This means that in the first day the weather is sunny with probability 0.25, cloudy with probability 0.5 and rainy with probability 0.25. For the second day we compute $\boldsymbol{\pi}^{(2)}$:

$$\boldsymbol{\pi}^{(2)} = \boldsymbol{\pi}^{(1)}P = \begin{bmatrix}0.25 & 0.5 & 0.25\end{bmatrix}\begin{bmatrix}0 & 0.50 & 0.50\\ 0.25 & 0.5 & 0.25\\ 0.25 & 0.25 & 0.5\end{bmatrix} \doteq \begin{bmatrix}0.19 & 0.44 & 0.37\end{bmatrix}.$$

We can simply predict the weather for days after.

Stationary distribution

In Example 5.5 (daily weather) let us continue to compute distribution vectors $\boldsymbol{\pi}^{(t)}$ for higher values of t, with $\boldsymbol{\pi}^{(t+1)} = \boldsymbol{\pi}^{(t)}P$ or equivalently $\boldsymbol{\pi}^{(t+1)} = \boldsymbol{\pi}^{(1)}P^t$. The results with initial vector $\boldsymbol{\pi}^{(0)} = [0, 1, 0]$ are (by rounding the final numbers to 4 decimal digits)

$$\boldsymbol{\pi}^{(1)} = \begin{bmatrix}0.25\\0.50\\0.25\end{bmatrix}^T,\ \boldsymbol{\pi}^{(2)} = \begin{bmatrix}0.1875\\0.4375\\0.3750\end{bmatrix}^T,\ \boldsymbol{\pi}^{(3)} = \begin{bmatrix}0.2031\\0.4063\\0.3906\end{bmatrix}^T,\ \ldots,\ \boldsymbol{\pi}^{(6)} = \begin{bmatrix}0.2000\\0.4001\\0.3999\end{bmatrix}^T,\ \boldsymbol{\pi}^{(7)} = \begin{bmatrix}0.2000\\0.4000\\0.4000\end{bmatrix}^T$$

The matrix P^7 is

$$P^7 \doteq \begin{bmatrix}0.2000 & 0.4000 & 0.4000\\ 0.2000 & 0.4000 & 0.4000\\ 0.2000 & 0.4000 & 0.4000\end{bmatrix}.$$

The same result with more or less number of iterations will obtained for other choices of initial distribution vector $\boldsymbol{\pi}^{(0)}$. The distribution of the Markov chain approaches the *stationary distribution* $\boldsymbol{\pi} = [0.2, 0.4, 0.4]$. In fact, all rows of P^t approaches this stationary distribution:

$$\lim_{t\to\infty} p_{ij}^{(t)} = \pi_j, \quad \forall i \in \mathcal{S} \tag{5.5}$$

with $\pi_j > 0$. This, on the other hand, means that if we set the initial distribution to

$$\boldsymbol{\pi}^{(0)} = [0.2, 0.4, 0.4],$$

then we can show that $\boldsymbol{\pi}^{(1)} = \boldsymbol{\pi}^{(0)}P = \boldsymbol{\pi}^{(0)} = [0.2, 0.4, 0.4]$, which means that $\boldsymbol{\pi}^{(0)}$ is also the distribution after one transition. Hence, it will be the distribution after two or more transitions.

Definition 5.4. The Markov chain $X_0, X_1, \ldots$ with transition matrix $P = (p_{ij})$ is called ergodic if the limits (5.5), i.e., π_j

1. exist for all $j \in \mathcal{S}$,
2. are positive ($\pi_j > 0$) and independent of $i \in \mathcal{S}$
3. form a probability vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_m)$, i.e., $\sum_{j\in\mathcal{S}} \pi_j = 1$.

If the limits (5.5) exist then $\lim_{t\to\infty} \boldsymbol{\pi}^{(t+1)} = \lim_{t\to\infty} \boldsymbol{\pi}^{(t)}P$ which means $\boldsymbol{\pi}P = \boldsymbol{\pi}$.

Definition 5.5 (Stationary Distribution)**.** Let P be the transition matrix for a Markov chain. A probability vector $\boldsymbol{\pi}$ that satisfies

$$\boldsymbol{\pi} P = \boldsymbol{\pi} \tag{5.6}$$

is called a *steady-state distribution* for the Markov chain.

From (5.6) we observe that for a finite Markov chain if the stationary distribution $\boldsymbol{\pi}$ exists then it is the eigenvector of P^T corresponds to eigenvalue 1, because

$$P^T \boldsymbol{\pi}^T = \boldsymbol{\pi}^T. \tag{5.7}$$

On the other hand, since the rows of P sum up to unity we have

$$P \boldsymbol{e}^T = \boldsymbol{e}^T$$

where $\boldsymbol{e} = [1, 1, \ldots, 1] \in \mathbb{R}^m$ which shows that 1 is an eigenvalue of P and thus an eigenvalue of P^T because eigenvalues of P and P^T are the same. This eigenvalue should correspond to at least one eigenvector for P^T that is $\boldsymbol{\pi}$ from (5.7). However, the uniqueness of this eigenvector is still not guaranteed.

Let us characterize the relations between states in the following way: For arbitrary but fixed states $i, j \in \mathcal{S}$ we say that the state j is accessible from state i if $p_{ij}^{(t)} > 0$ for some $t \geqslant 0$, we say that i is accessible from (or leads to) state j and write $i \to j$. We say that i and j *communicate* if $i \to j$ and $j \to i$, and write $i \leftrightarrow j$. Using the relation '$i \leftrightarrow j$', we can divide the state space $\mathcal{S}$ into equivalence classes such that all the states in an equivalence class communicate with each other but not with any state outside that class. If there is only one class (i.e., $\mathcal{S}$ itself), the Markov chain is said to be *irreducible*. Besides irreducibility we need a second property of the transition probabilities, namely aperiodicity. The period d_i of the state $i \in \mathcal{S}$ is given by

$$d_i = \gcd\{t \geq 1 : p_{ii}^{(t)} > 0\}$$

where "gcd" denotes the greatest common divisor. We define $d_i = \infty$ if $p_{ii}^{(t)} = 0$ for all $t \geq 1$. A state $i \in \mathcal{S}$ is said to be aperiodic if $d_i = 1$. The Markov chain $\{X_t\}$ and its transition matrix $P = (p_{ij})$ are called *aperiodic* if all states of $\{X_t\}$ are aperiodic. We can show that the periods d_i and d_j coincide if the states i, j belong to the same equivalence class of communicating states. Thus, if the Markov chain $\{X_t\}$ is irreducible then all its states have the same period. Here we give the statements of two fundamental theorems without proofs. For more details refer to [**Rubinnstein-Kroese:2017**].

Theorem 5.6. The Markov chain $X_0, X_1, \ldots$ is ergodic if and only if it is irreducible and aperiodic.

Theorem 5.7. For an irreducible and aperiodic Markov chain $X_0, X_1, \ldots$ (ergodic Markov chain) with transition matrix P, the stationary distribution $\boldsymbol{\pi}$ is *uniquely* determined by

solving the eigenvalue problem (5.6). The eigenvalue $\lambda_1 = 1$ is the dominant eigenvalue and $|\lambda_k| < |\lambda_1| = 1$ for $\lambda = 2, 3, \ldots, m$.

5.2 Random walk on the integers

Assume that your initial state is $X_0 = 0$ and at each time t you walk by steplength 1 either to the right or to the left of your current position X_t with probabilities p and q, respectively, where $p \in (0, 1)$ is a real number and $q = 1 - p$. The process X_t is called a simple random walk which has the state space $\mathcal{S} = \mathbb{Z}$ (set of integer numbers) and a transition graph shown in Figure 16.

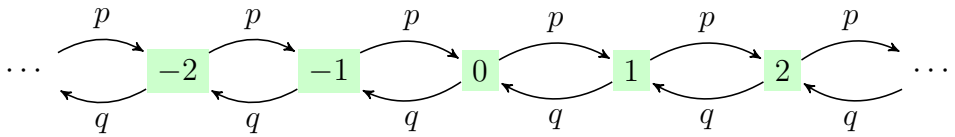


Figure 16: Transition graph for random walk on integers

The random walk X_t can be characterized as

$$X_{t+1} = X_t + \tilde{B}_t, \quad t = 0, 1, 2, \ldots,$$

where $\tilde{B}_t$ has a Bernoulli distribution with state $\{-1, 1\}$ and probabilities q and p. Indeed $\tilde{B}_t = 2B_t - 1$ where $B \sim \mathcal{B}er(p)$, the standard Bernoulli distribution. The above relation shows that X_{t+1} depends solely on X_t, i.e. the random walk is a Markov process. It is also a Markov chain because the index set for t is discrete. We also observe that

$$X_t = \sum_{j=0}^{t-1} \tilde{B}_j,$$

which means that the distribution of X_t is binomial[2]. Since $\mathbb{E}(\tilde{B}_j) = 2p - 1$ and $\mathrm{Var}(\tilde{B}_j) = 4p(1-p)$, we simply have

$$\mathbb{E}(X_t) = t(2p - 1), \quad \mathrm{Var}(X_t) = 4tp(1 - p).$$

For special case $p = q = 1/2$ we have $\mathbb{E}(X_t) = 0$ and $\mathrm{Var}(X_t) = t$.

The transition matrix of the random walk process has the form

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

Since X starts at 0, i.e., $\mathbb{P}(X_0 = 0) = 1$ and $\mathbb{P}(X_0 = j) = 0$ for all $j \in \mathbb{Z} \setminus \{0\}$, we have

$$\boldsymbol{\pi}^{(0)} = [\cdots, 0, 0, 1, 0, 0, \cdots].$$

[2]The sum of Bernoulli distributions is a binomial distribution.

Then $\pi^{(1)} = \pi^{(0)}P$ has the form

$$\boldsymbol{\pi}^{(1)} = [\cdots, 0, q, 0, p, 0, \cdots],$$

which means that after one step the chain moves to right with probability p and to left with probability q.

5.3 Gaussian processes

Gaussian processes are generalizations of multivariate normal distributions. If you are not familiar with multivariate (normal) distributions, see sections A.5-A.7 in the Appendix.

Definition 5.8. The stochastic process $\{X_t,\, t \in \mathcal{T}\}$ is called a *Gaussian process* if all its finite dimensional distributions are normal (Gaussian). In the other words, $\{X_t,\, t \in \mathcal{T}\}$ is a Gaussian process if for any choice of n and $t_1, t_2, \ldots, t_n \in \mathcal{T}$ we have

$$(X_{t_1}, X_{t_2}, \ldots, X_{t_n}) \sim \mathcal{N}(\mu, \Sigma)$$

for some expectation vector μ and covariance matrix Σ both depend on the choice of $t_1, \ldots, t_n$.

Equivalently, $\{X_t,\, t \in \mathcal{T}\}$ is a Gaussian process if any linear combination

$$\sum_{k=1}^{n} c_k X_{t_k}$$

has a normal distribution. A Gaussian process is fully determined by its expectation function $\mu(t) = \mathbb{E}(X_t)$ for $t \in \mathcal{T}$ and covariance function $\Sigma(s,t) = \mathrm{Cov}(X_s, X_t)$ for $s, t \in \mathcal{T}$.

An important Gaussian process is the *Wiener process* or the standard *Brownian motion*, which can also be considered as a continuous version of the random walk on the integers.

Definition 5.9. A Gaussian process $\{W_t,\, t \in \mathcal{T}\}$ with $\mu(t) = 0$ for all $t \in \mathcal{T}$ and $\Sigma(s,t) = s$ for all $0 \leqslant s \leqslant t$ is called a Wiener process.

We can prove that, a Wiener process is a Markov process with a continuous sample path that is nowhere differentiable. Moreover, in the Wiener process the increments $W_t - W_s$ on intervals $[s,t]$ are independent and normally distributed. More precisely, from the definition and for two choices $s, t \in \mathcal{T}$ with $0 \leqslant s \leqslant t$ we have

$$\begin{bmatrix} W_s \\ W_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} s & s \\ s & t \end{bmatrix} \right).$$

The Cholesky factorization of the covariance matrix is

$$\Sigma = \begin{bmatrix} s & s \\ s & t \end{bmatrix} = \begin{bmatrix} \sqrt{s} & 0 \\ \sqrt{s} & \sqrt{t-s} \end{bmatrix} \begin{bmatrix} \sqrt{s} & \sqrt{s} \\ 0 & \sqrt{t-s} \end{bmatrix} =: BB^T$$

which results in

$$\begin{bmatrix} W_s \\ W_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \sqrt{s} & 0 \\ \sqrt{s} & \sqrt{t-s} \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$$

where Z_1 and Z_2 are two univariate standard normal distributions[3]. Solving the system gives $W_s = \sqrt{s}Z_1$ and $W_t = W_s + \sqrt{t-s}Z_2$. The later shows

$$W_t - W_s \sim \mathcal{N}(0, t-s), \quad \text{for all } t \geqslant s \geqslant 0. \tag{5.8}$$

This proves that in the Wiener process the increments $W_t - W_s$ on intervals $[s, t]$ are independent and normally distributed.

6 Stochastic process generation

In this section, we provide a brief overview of algorithms for generating a few standard stochastic processes. However, stochastic processes are not limited to the standard examples presented here. For any specific stochastic model, users can use standard methods for sampling random points and random processes to generate a specific stochastic process that is a solution to their model. In section 7 we will see an example.

6.1 Generating Markov chains

Assume that $X = \{X_0, X_1, \ldots, X_n\}$ is the first $n+1$ random variables of a finite Markov chain with initial distribution $\boldsymbol{\pi}^{(0)}$, state space $\mathcal{S} = \{1, 2, \ldots, m\}$, and transition matrix P. We follow the simulation process given at the beginning of section A.9 for dependent variables by the use of conditional distributions. We first generate X_0 from distribution $\boldsymbol{\pi}^{(0)}$. If $X_0 = i$ is generated, we then generate X_1 from the conditional distribution of X_1 given $X_0 = i$. In the other words, we generate X_1 from the i-th row of P. Let $X_1 = j$ be generated. From here on, we use the Markov property, so we generate X_2 from the conditional distribution of X_2 given $X_1 = j$, i.e. we generate X_2 from the j-th row of P. This process is continued until X_n is generated.

In the Python function below, the input variables `InitDist`, `TransMat` and `ChainLen` play the roles of $\boldsymbol{\pi}^{(0)}$, P and n, respectively. The outputs are the integer vector (state vector) `X` of size n, and the stationary distribution `StationDist` of length m. The stationary distribution is approximated by m iterations of the *power method* for the dominant eigenvector of P. Note that, the `RandDisct` function (the discrete random variable generator) is called to generate random numbers from the rows of P.

```
def MarkovChainGen(InitDist, TransMat, ChainLen):
    # This function generates a Markov chain of length 'ChainLen' from
    #   Transition matrix 'TransMat' with initial distribution 'InitDist'
    m = len(InitDist)
    states = range(m)
    X = np.zeros(ChainLen)
```

[3] If $W \sim \mathcal{N}(\mu, \Sigma)$ then $W = \mu + BZ$ where $Z \sim \mathcal{N}(0, I)$ and B is the Cholesky factor of Σ.

```
    p = StationDist = InitDist
    for j in range(ChainLen):
        i = RandDisct(states,p,1)[0]
        X[j] = i
        p = TransMat[i,:]
        StationDist = TransMat@StationDist
    return X, StationDist
```

Example 6.1. Consider again the weather forecasting model in Example 5.4. We start with the initial state of sunny on the day 0. We want to estimate the probability of a rainy day on the fifth day. We can generate multiple weather sequences of length 5 (Markov chains of length 5) and use Monte Carlo to estimate the probability of a rainy day on the fifth day. The code is given below.

```
# Transition matrix
P = np.array([[0.00, 0.50, 0.50],  # From sunny to (sunny, cloudy, rainy)
              [0.25, 0.50, 0.25],  # From cloudy to (sunny, cloudy, rainy)
              [0.25, 0.25, 0.50]]) # From rainy to (sunny, cloudy, rainy)
# Map state names to indices for the matrix
StateMap = {"sunny": 0, "cloudy": 1, "rainy": 2}
InitDist = [1,0,0]  # [sunny, cloudy, rainy]
RainyCount = 0
N = 1000
for i in range(N):    # Run N simulations
    X, S = MarkovChainGen([1,0,0], P, 5)
    if X[-1] == StateMap["rainy"]: # Check if the fifth day ended in rainy
        RainyCount += 1

PrRainy = RainyCount / N  # Estimate probability
print('Estimated probability of a rainy fifth day = ', PrRainy)
```

An execution gives the answer `0.402` which shows that by about 40% probability the fifth day is rainy.

6.2 Random walk on the integers

Random walk on the integer is Markov chain as it was described in section 5.2. Thus, the `MarkovChainGen` function can be used to generated this process. However, a simpler algorithm

without forming the transition matrix P is based on the relation $X_{t+1} = X_t + \tilde{B}_t$.

```
def RandWalkGen(p, X0, t):
    B = np.append(X0,2*RandBer(p,t)-1)
    X = np.cumsum(B)
    return X
```

A typical sample path for the case $p = q = 1/2$, $X_0 = 0$ and $t = 100$ is plotted in Figure 17. The horizontal axis represents the time index $t = 0, 1, 2, \ldots, 100$ and the vertical axis the states $j \in \mathbb{Z}$.

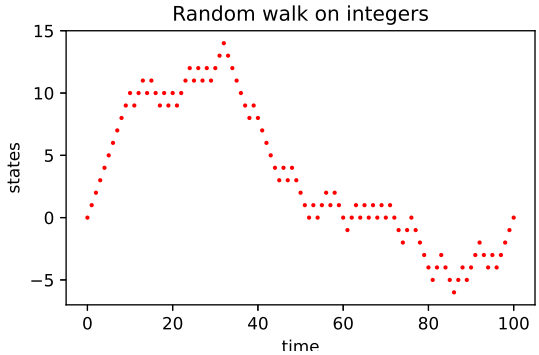


Figure 17: A random walk path on the integers with $p = 1/2$.

Example 6.2 (gambler's ruin)**.** Suppose a gambler starts with a certain amount of money, say $\$K$, and in each round, either wins or loses a fixed amount, say \$1, with probabilities p and $1 - p$, respectively. The game continues until the gambler either reaches a target amount $\$T$ or loses all the money (being *ruined*). The target amount T is an *absorbing state* on the right side and the amount 0 is an absorbing state on the left side.
This scenario is a random walk. The goal is to determine the probability that the gambler reaches the target amount T before being ruined. We generate many sample paths and use Monte Carlo to estimate this probability. The simulation should track whether the gambler ends up with $X_t = 0$ or $X_t = T$ in multiple paths. By averaging the outcomes, we estimate the probability of ruin. The code is given below for some values of parameters.

```
T = 100   # target amount
K = 30    # initial money
p = 0.5   # probability of winning each round
N = 1000  # number of MC simulations
RuinCount = 0
for _ in range(N):
    money = K
```

```
    while money > 0 and money < T:
        B = 2*RandBer(p,1)-1
        money += B
    if money == 0:
        RuinCount += 1
PrRuin = RuinCount / N  # Estimated probability of ruin
print('Estimated Probability of Ruin = ', PrRuin)
```

An execution shows the probability `0.703`. This problem has indeed an exact solution

$$\mathbb{P}(\text{ruin}) = \begin{cases} \frac{1-\left(\frac{q}{p}\right)^k}{1-\left(\frac{q}{p}\right)^T} & \text{if } p \neq q, \\ \frac{T-k}{T} & \text{if } p = q = \frac{1}{2}, \end{cases}$$

which can be used to assess the accuracy of the above Monte Carlo algorithm.

A random walk on integers can model several real-world scenarios. In a financial modeling, a random walk can be used to simulate investment portfolios, where each step represents a gain or loss in asset value. In population dynamics, a random walk can model population survival, where each step could represent birth or death events. In a biological model, a random walk can describe gene mutations, where each step indicates a genetic drift towards different traits or alleles.

The random walk can be extended to d dimensions by replacing the Bernoulli distribution with a general discrete distribution for updates. We indeed have

$$X_{t+1} = X_t + D_t$$

where $D_t \sim \mathcal{DD}([x_1, \ldots, x_{2d}], [p_1, \ldots, p_{2d}])$. As an example in 2 dimensions, we have $x_1 = [-1, 0]$, $x_2 = [1, 0]$, $x_3 = [0, 1]$, and $x_4 = [0, -1]$, which means that at each time step the process moves to left, right, up or down with probabilities p_1, p_2, p_3 and p_4 respectively.

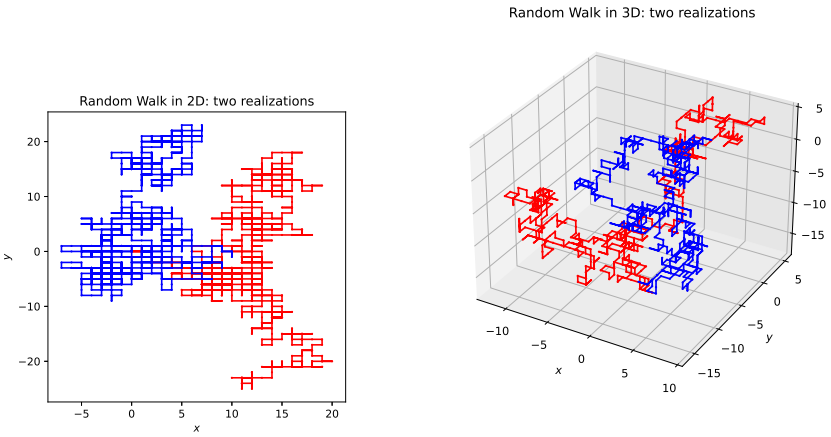


Figure 18: Random walk realizations in 2 and 3 dimensions.

Two random walk realizations for $X_0 = [0, 0]$ and $p_k = 1/4$ are shown on the left side of Figure 18. The same extension applies in 3 dimensions with 6 possible different directions for a new update. A couple of realizations are shown on the right side of Figure 18 for $X_0 = [0, 0, 0]$ and $p_k = 1/6$. In both 2 and 3 dimensions, the code is left as an exercise to the reader.

6.3 Generating Gaussian processes

In a Gaussian process $\{X_t : t \in \mathcal{T}\}$, each finite dimensional vector $(X_{t_1}, \ldots, X_{t_n})$ has a multivariate normal distribution. This means that any multivariate normal sampler can be used to generate realizations of a Gaussian process at prescribed times $t_1, \ldots, t_n$ provided that the mean vector $\mu = (\mu(t_1), \ldots, \mu(t_n))$ and the covariance matrix $\Sigma = (\Sigma(t_k, t_j))$ for $k, j = 1, 2 \ldots, n$ are given.

For a Wiener process (or Brownian motion) $\{W_t : t \in \mathcal{T}\}$, as an special case, the algorithm can be simplified. For this process we have $\mu(t) = 0$ and $\Sigma(s, t) = s$ for $0 \leqslant s \leqslant t$. For given times $t_1, \ldots, t_n$ with $0 < t_1 < t_2 < \cdots < t_n$, the covariance matrix for variable $(W_{t_1}, \ldots, W_{t_n})$ has the form

$$\Sigma = \begin{bmatrix} t_1 & t_1 & t_1 & \cdots & t_1 \\ t_1 & t_2 & t_2 & \cdots & t_2 \\ t_1 & t_2 & t_3 & \cdots & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \cdots & t_n \end{bmatrix}.$$

The Cholesky factor for Σ is

$$B = \begin{bmatrix} \sqrt{t_1} & 0 & 0 & \cdots & 0 \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & 0 & \cdots & 0 \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \sqrt{t_3 - t_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \sqrt{t_3 - t_2} & \cdots & \sqrt{t_n - t_{n-1}} \end{bmatrix}.$$

Then $W = \mu + BZ = 0 + BZ$ where Z is a vector of iid random variables with distributions $\mathcal{N}(0, 1)$, i.e.,

$$W = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_n \end{bmatrix} = B \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} = \begin{bmatrix} W_0 + \sqrt{t_1 - t_0} Z_1 \\ W_1 + \sqrt{t_2 - t_1} Z_2 \\ \vdots \\ W_{n-1} + \sqrt{t_n - t_{n-1}} Z_n \end{bmatrix}$$

for $t_0 = 0$ and $W_0 = 0$. This means that

$$W_{k+1} = W_k + \sqrt{t_{k+1} - t_k}\, Z_{k+1}, \quad Z_{k+1} \sim \mathcal{N}(0, 1), \quad W_0 = 0.$$

The Python function is given below.

```
def BrownianMotionGen(t_vec, dim):
    # This functions generates a dim-dimensional Brownian motion on
    #    time samples t_vec = [t_1, t_2, ..., t_n]
    n = len(t_vec)
    W = np.zeros([dim,n])
    for k in range(n-1):
        Z = np.random.normal(0,1,dim)
        W[:,k+1] = W[:,k] + np.sqrt(t_vec[k+1]-t_vec[k])*Z
    return W
```

However, the function works also for higher dimensions. In higher dimensions each step of the process updates by a vector whose components are drawn from a multivariate normal distribution. We note that in higher dimensions the term *Brownian motion* is commonly used while in the 1-dimensional case both terms *Wiener process* and *Brownian motion* are used.

In Figure 19 two sample paths of the 1D, 2D and 3D Brownian motions in time interval $[0, 1]$ are shown at times $t_k = k\Delta t$ for $\Delta t = 10^{-3}$.

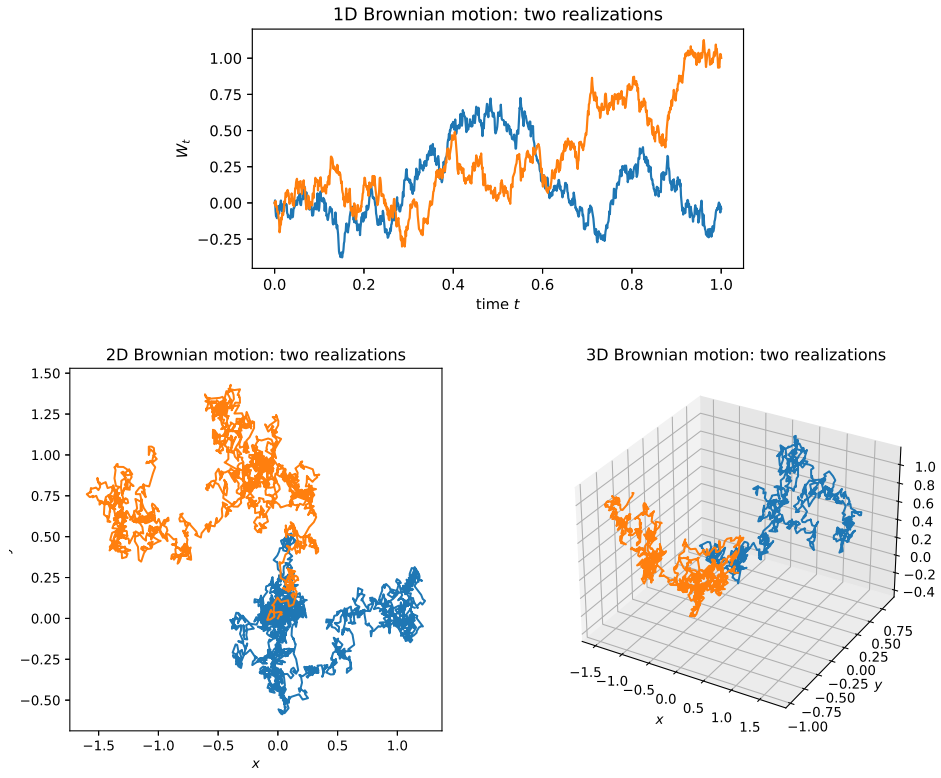


Figure 19: Two realizations of the Wiener process on time interval $[0, 1]$ in 1D (top), 2D (down-left) and 3D (down-right).

.

Example 6.3 (Brownian particles). The motion of particles suspended in a medium, such as liquid or gas, was first observed by the Scottish botanist Robert Brown in 1827. While examining pollen from the plant *Clarkia pulchella* under a microscope, he noticed the erratic

movement of pollen grains in water where large number of water molecules "push" the particles in different directions. This particle motion was later understood to be a stochastic process, now referred to as Brownian motion, as we discussed it above.

Now, consider the movement of a single Brownian particle in the water that is confined to a fixed cubic box, say $[-1, 1]^3$. Our goal is to estimate the *expected time* required for a Brownian particle, initially positioned at the center of the box, to hit one of the box's boundaries.

To solve this problem, we can use the Monte Carlo method. We simulate a large number of 3D Brownian motion paths, say 10^4 realizations. For each simulated path, we record the time taken for the particle to first reach one of the box boundaries. Then we compute the mean of these recorded times to estimate the expected hitting time. The following code provides an implementation for this Monte Carlo simulation approach.

```
import numpy as np
dt, N = 0.005, 10000
HitTime = np.empty(N)
for j in range(N):
    k = 0
    W = np.array([0,0,0])
    while True:
      W = W + np.sqrt(dt)*np.random.normal(0,1,3)
      if any(abs(W) >= 1):
        break
      k += 1
    HitTime[j] = dt*k
HitTime_mean = np.mean(HitTime)
HitTime_std = np.std(HitTime)
err = 1.96*HitTime_std/np.sqrt(N)
print('Expected Hitting Time = ', HitTime_mean)
print('Error with 95% probability = ', err)
```

In the code, the hitting time is computed by the product of time step Δt and number of steps until the boundary is hit. We choose a small time step $\Delta t = 0.005$ and a large value $N = 10^4$ (number of simulations). Finally we computed both the mean and the error of estimation with 95% probability using the formula $1.96s/\sqrt{N}$ where s is the sample standard deviation. The output for a run reported below.

```
Expected Hitting Time =  0.4856
Error with 95% probability =  0.0061
```

Another class of processes are *diffusion processes* which are Markov with a continuous time parameter and continuous sample pathes, like as Wiener processes. Wiener processes form a basis for defining and generating diffusion processes. In fact, a diffusion process is defined as the solution of the following *stochastic differential equation* (SDE)

$$\mathrm{d}X_t = a(t, X_t)\mathrm{d}t + b(t, X_t)\mathrm{d}W_t \tag{6.1}$$

where $\{W_t : t \geqslant 0\}$ is Wiener process and $a(t, x)$ and $b(t, x)$ are some deterministic functions, usually refereed to as *drift* and *diffusion* coefficients, respectively. The resulting process for spacial case $a(t, x) = \mu$ and $b(t, x) = \sigma$ is obtained as

$$X_t = \mu t + \sigma W_t$$

and is called a *Brownian motion*. The more special case with $\mu = 0$ and $\sigma = 1$ gives the Wiener process or the standard Brownian motion $X_t = W_t$. The case $a(t, x) = \mu x$ and $b(t, x) = \sigma x$ results in the *geometric Brownian motion*.

For generating a diffusion process (approximately) we can use the explicit *Euler–Maruyama method*[4] to discretize (6.1) as

$$Y_{k+1} = Y_k + a(t_k, Y_k)\Delta t + b(t_k, Y_k)\sqrt{\Delta t}Z_{k+1}, \quad k = 0, 1, 2, \ldots,$$

where Δt is a time step, $t_k = k\Delta t$, and $Z_1, Z_2, \ldots$ are independent random variables with $\mathcal{N}(0, 1)$ distributions. The process $\{Y_k, k = 0, 1, \ldots\}$ approximates the exact process $\{X_t, t \geqslant 0\}$ in the sense that $Y_k \approx X_{k\Delta t}$. The initial variable Y_0 should be generated from the distribution of X_0. The Python function is given below.

```
def DiffusionProcessGen(drift, diffusion, tspan, dt, X0, *args):
    # This function generates a diffusion process by solving the SDE
    #       dX_t = a(t,X_t) dt + b(t,X_t)dW_t,   X_0 = X0
    # for drift coefficient a(t,x) and diffusion coefficient b(t,x)
    # on interval tspan = [t_0, t_end] at equidistance times t_0, t_1, ...,t_n
    # using the explicit Euler-Maruyama method
    t = tspan[0]
    Y = {}
    Y[0] = X0; j = 0;
    while t <= tspan[1]:
        dW = np.sqrt(dt)*np.random.randn()
        Y[j+1] = Y[j] + drift(t,Y[j],*args)*dt + diffusion(t,Y[j],*args)*dW
        t += dt; j += 1
    Y = [Y[i] for i in range(len(Y))]
    return Y
```

[4]The explicit Euler–Maruyama method is the stochastic equivalent of the explicit Euler method for solving SDEs.

Example 6.4. One application of Brownian motions in combination with the Monte Carlo method is in the pricing of financial options. Here we give an example for pricing a *European Call Option*.

In finance, a European call option is a contract that gives the holder the right, but not the obligation, to buy a stock at a fixed price (the strike price K) at a specified future date (the expiration time T). The *Black-Scholes* model assumes that the stock price follows the SDE

$$\mathrm{d}S_t = \mu S_t \,\mathrm{d}t + \sigma S_t \,\mathrm{d}W_t,$$

where S_t is the stock price at time t (in year), μ is the drift term representing the average rate of return, σ is the volatility (standard deviation of returns), and W_t is a Wiener process. As we observe, the stock price S_t is indeed a geometric Brownian motion.

The payoff of the call option at the final time is

$$C_T = \max(S_T - K, 0)$$

for the given strike price K. We know from Feynman-Kac that the value of the call option at earlier times $t < T$ is given by

$$C_t = \mathbb{E}(e^{-r(T-t)} C_T | S_t)$$

where r is the risk-free interest rate. This expectation is taken under the appropriate risk-neutral measure, which sets the drift μ equal to the risk-free rate r. The option price at time $t = 0$ (generally representing the present year) for a given initial stock price S_0 is

$$C_0 = \mathbb{E}(e^{-rT} C_T) = e^{-rT} \mathbb{E}(C_T).$$

To apply the Monte Carlo method to estimate C_0, we generate N (a large number) simulation paths for S_t up to time $t = T$ and for each path we calculate the payoff at expiration, i.e. C_T. Finally we compute the mean of C_T and multiply it by e^{-rT}.

As an example, we assume that the expiration time is $T = 0.5$ a year, the current asset price is $S_0 = 102$, the volatility is $\sigma = 30\%$, the interest rate is $r = 4\%$, and the strike price is $K = 100$. We use $\Delta t = 0.001$ and generates $N = 10^4$ Monte Carlo simulations. We also estimate the standard error.

```
r = 0.04        # risk-free interest rate
mu = r          # drift coefficient (average rate of return) and interest
sigma = 0.3     # volatility
S0 = 102        # initial stock price
K = 100         # strike price
T = 0.5         # expiration time
dt = 0.001      # steplength for Euler-Maruyama method
def drift(t,x,*args):      # drift
    m = args[0]
    return m*x
```

```
def diffusion(t,x,*args):  # diffusion
    s = args[1]
    return s*x
N = 10**4             # number of MC simulations
CT = np.empty(N)
for j in range(N):
    S = DiffusionProcessGen(drift, diffusion, [0,T], dt, S0, mu, sigma)
    ST = S[-1]
    CT[j] = max(ST-K , 0)
C0 = np.exp(-r*T)*np.mean(CT)
std = np.std(np.exp(-r*T)*CT)
err = 1.96*std/np.sqrt(N)
print("The call value is {0}' +/- {1} with 95% probability".format(V0, err))
```

An execution gives the following output:

```
The call value is 10.0314 +/- 0.2898 with 95% probability
```

In the Monte Carlo loop, we generated N paths of a geometric Brownian motion with an initial value $S_0 = 102$. Ten trajectories are shown on the left side of Figure 20. Note that only the final value of each path, S_T, was used to compute the payoff of the option. The histogram of random variable S_T is also shown on the right-hand side of figure 20 which shows that the distribution of S_T is a log-normal distribution. This is consistent with the theoretical property of geometric Brownian motion. Note that a random variable X is said to have a log-normal distribution if $\log(X)$ has a normal distribution.

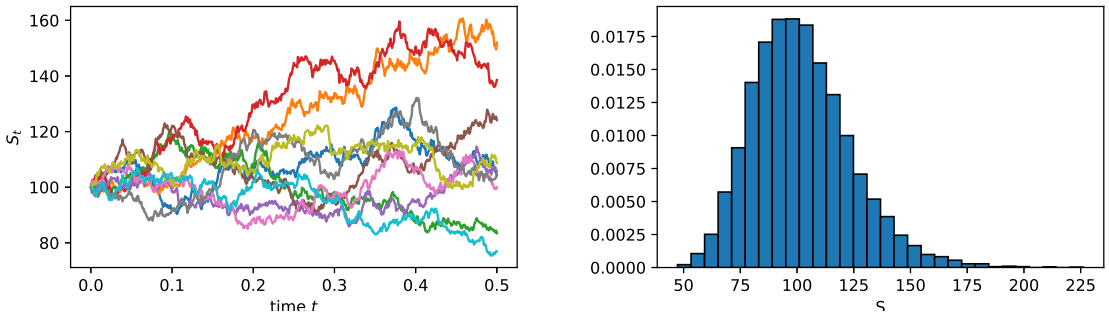


Figure 20: Ten realizations of the geometric Brownian motion S_t for $t \in [0, 0.5]$ (left), the histogram of the variable S_T for $T = 0.5$ (right)

7 Stochastic Simulation Algorithm (SSA)

The Stochastic Simulation Algorithm (SSA), also known as the *Gillespie algorithm*, proposed by Daniel T. Gillespie in 1977[5], is a powerful method employed in systems biology to predict the behavior of complex biological systems. Other areas of application are in epidemiology and ecology.

7.1 Simulation of a simple epidemic model

To illustrate the SSA algorithm, we consider a susceptible-infected-recovered (SIR) model. Such models describe the spread of a virus (for example influenza, covid, etc.) within a population. The population is divided into three groups (states): susceptible individuals, infected individuals, and recovered individuals. Each state is represented by the variables S, I, and R, respectively. The relationships between these states can be depicted in a flow diagram (transition graph), where transitions occur based on specific rates. See Figure 21.

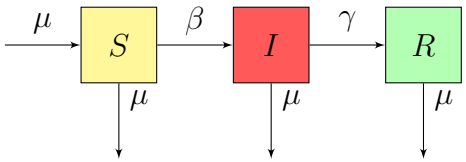


Figure 21: The transition graph of a simple SIR model.

The model has some parameters: μ is the birth and death rate, β is the infection rate, and γ is the recovery rate. The unit of S, I, and R is individual, and the unit of rates is one over time unit, for example $\frac{1}{\text{day}}$.

To simplify the mathematical formulation, we assume that the variables $S(t)$, $I(t)$, and $R(t)$ are continuous functions over time t, and the simulation is valid in a defined interval $[0, t_{\text{final}}]$. Additionally, it is assumed that the rates of birth and death are equal, and all newborns are susceptible. The model does not account for pathogen-induced mortality (death because of the virus), and it is assumed that recovered individuals remain immune throughout the period of the epidemic. Furthermore, the infection rate depends on the ratio of infected individuals to the total population, expressed as $\beta\frac{I}{N}$. This assumption is quite reasonable, as a higher number of infected individuals increases interactions between the susceptible group and the infected group which leads to an elevated infection rate over time.

Given the assumption above, we can write a system of ordinary differential equations (ODEs) to describe the dynamics of the SIR model. We denote the total population at time t by

$$N(t) = S(t) + I(t) + R(t).$$

The differential equations governing the dynamics are as follows:

[5]Daniel T. Gillespie, *Exact Stochastic Simulation of Coupled Chemical Reactions*, The Journal of Physical Chemistry, Vol. 81, No. 25, 1977.

$$\frac{dS}{dt} = \mu N - \mu S - \beta \frac{I}{N} S$$
$$\frac{dI}{dt} = \beta \frac{I}{N} S - \mu I - \gamma I$$
$$\frac{dR}{dt} = \gamma I - \mu R$$

with initial conditions

$$S(0) = S_0, \quad I(0) = I_0, \quad R(0) = R_0.$$

This initial value problem is a deterministic model for the given phenomenon. To solve this system of equations, we can employ a deterministic numerical methods (an ODE solver) such as a Runge-Kutta method. Here we call the `RK45` from the `scipy.integrate.solve_ivp` module.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

mu, bet, gam = 1e-4, 0.25, 0.05   # rates
Initial = [198,2,0]               #   [S(0), I(0) R(0)]
FinalTime = 120                   # final time of simulation

def ODEfun(t,y):
    yprime = np.zeros(3);
    S,I,R = y
    N = np.sum(y)
    yprime[0] = mu*N - bet*S*I/N-mu*S
    yprime[1] = bet*S*I/N -(mu+gam)*I
    yprime[2] = gam*I - mu*R
    return yprime

teval = np.linspace(0, FinalTime,500)
sol = solve_ivp(ODEfun, [0,FinalTime], Initial, t_eval = teval)

plt.figure(figsize = (6, 4))
plt.plot(sol.t,sol.y[0],linestyle = 'solid', color='blue', label = '$S$')
plt.plot(sol.t,sol.y[1],linestyle = 'solid', color='red', label = '$I$')
plt.plot(sol.t,sol.y[2],linestyle = 'solid', color='green', label = '$R$')
plt.xlabel('time $t$'); plt.ylabel('Individuals')
plt.title('Deterministic solution using RK45')
plt.legend(loc='center right')
```

The results are given in Figure 22 which illustrates the dynamics of the susceptible, infected, and recovered populations over time. The number of infected individuals increases from initially two to approximately 125 and reaches its peak around the 20th day. Following this peak, the epidemic enters a decline phase, with the number of infections decreasing rapidly until the epidemic eventually comes to an end.

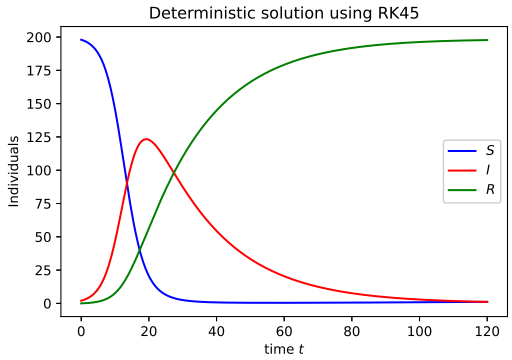


Figure 22: Solution of SIR model using `RK45`.

While the above deterministic model provides valuable insights, it overlooks the discrete nature of populations and the inherent uncertainties present in the model. A stochastic approach reformulates the model by replacing continuous variables $S(t)$, $I(t)$, and $R(t)$ with discrete values. We define a series of reactions (processes) with their associated *propensity functions* which are indeed the number of individuals moving into or out of the groups:

$$\begin{aligned}
&1. \quad \emptyset \longrightarrow S && w_1 = \mu N \\
&2. \quad S \longrightarrow I && w_2 = \beta \tfrac{I}{N} S \\
&3. \quad I \longrightarrow R && w_3 = \gamma I \\
&4. \quad S \longrightarrow \emptyset && w_4 = \mu S \\
&5. \quad I \longrightarrow \emptyset && w_5 = \mu I \\
&6. \quad R \longrightarrow \emptyset && w_6 = \mu R
\end{aligned}$$

The reactions include transitions from susceptible to infected individuals, from infected to recovered individuals, and the natural birth and death processes affecting all the groups. The propensity functions represent the tendency (or likelihood) of each reaction to occur. We now assume a state vector

$$\boldsymbol{y}(t) = (y_1(t), \ldots, y_n(t))$$

containing the state variables. For instance, for the SIR model we have $n = 3$ states and $m = 6$ reactions, with the propensity functions corresponding to each reaction:

$$\boldsymbol{y} = (S, I, R),$$

$$w_1 = \mu N,\ w_2 = \beta \frac{I}{N} S,\ w_3 = \gamma I,\ w_4 = \mu S,\ w_5 = \mu I,\ w_6 = \mu R.$$

Also we compute the *total propensity* function

$$a(\boldsymbol{y}) = \sum_{j=1}^{m} w_j(\boldsymbol{y})$$

which represents the overall tendency of the system to evolve and undergo changes. The positive propensity functions w_j represent the relative likelihood of each reaction occurring compared to the others. To convert these into probabilities, we define

$$p_j := \frac{w_j}{a}, \quad j = 1, \ldots, m$$

where a is the total propensity. This allows us to form the following discrete distribution table

reaction j	1	2	...	m
probability p_j	p_1	p_2	...	p_m

which tells us which reactions are more likely to occur and change the system states.

The basic assumption in the SSA is that, between time t and $t + \tau$, exactly one reaction occurs and causes a change in the system's states by either 0, $+1$, or -1. Therefore, at any given time t, the SSA includes three steps:

1. *when* the next reaction will occur (i.e. the value of τ)
2. *which* reaction will occur
3. *Updating the state* of the system

The waiting time τ for the next reaction is sampled from an exponential distribution with rate parameter $\lambda = a(\boldsymbol{y})$:

$$when \sim \mathcal{E}xp\left(a(\boldsymbol{y})\right).$$

A higher value of a (a higher total propensity) corresponds to a shorter expected waiting time between events.

Which reaction? The reactions with higher probabilities are more likely to occur. To determine which reaction takes place, we sample from the discrete distribution based on the probabilities p_j defined earlier:

$$which \sim \mathcal{DD}\Big([1, 2, \ldots, m], [p_1, \ldots, p_m]\Big).$$

Once the reaction is determined, the final step is to update the system's state. We use the *state-change* vectors $\boldsymbol{v}_j$ (also called the *stoichiometry* vectors), which represent how each reaction alters the state of the system. For example, in the SIR model, the vectors are as follows:

$$\begin{aligned}
&1. \quad \emptyset \longrightarrow S, && \boldsymbol{v}_1 = [\,1, \quad 0, \quad 0] \\
&2. \quad S \longrightarrow I, && \boldsymbol{v}_2 = [-1, \quad 1, \quad 0] \\
&3. \quad I \longrightarrow R, && \boldsymbol{v}_3 = [0, \; -1, \quad 1] \\
&4. \quad S \longrightarrow \emptyset, && \boldsymbol{v}_4 = [-1, \quad 0, \; 0] \\
&5. \quad I \longrightarrow \emptyset, && \boldsymbol{v}_5 = [\,0, \; -1, \; 0] \\
&6. \quad R \longrightarrow \emptyset, && \boldsymbol{v}_6 = [\,0, \quad 0, \; -1]
\end{aligned}$$

These vectors describe how the number of susceptible, infected, or recovered individuals changes due to each specific reaction.

In the *which* phase if the reaction index k is sampled, then the state is updated based on

the corresponding state-change vector $\boldsymbol{v}_k$, i.e.,

$$\boldsymbol{y}(t+\tau) = \boldsymbol{y}(t) + \boldsymbol{v}_k.$$

Considering all these together, the Gillespie algorithm is presented below.

Algorithm 5 Gillespie algorithm (SSA)

Require: Initial state $\boldsymbol{y} = \boldsymbol{y}_0$, final time t_{final}, propensity functions $w_1, \ldots, w_m$ and state change vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$
Ensure: State vector $\boldsymbol{y}(t)$ at final time $t = t_{final}$
  $t \leftarrow 0$
  while $t \leq t_{final}$ **do**
    Compute $a(\boldsymbol{y}) = w_1(\boldsymbol{y}) + \cdots + w_m(\boldsymbol{y})$ and $p_j(\boldsymbol{y}) = w_j(\boldsymbol{y})/a(\boldsymbol{y})$
    Generate $\tau \sim \mathcal{E}xp(a(\boldsymbol{y}))$
    Generate $k \sim \mathcal{DD}\Big([1, \ldots, m], [p_1, \ldots, p_m]\Big)$
    Update $t \leftarrow t + \tau$
    Update $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{v}_k$
  end while

7.2 Python implementation

In this section, we implement the SSA using Python. At each time step, the code calls the `RandExp` and `RandDisct` functions (Section 3) to sample the steplength and determine which reaction occurs.

```
##  Gillespie algorithm (SSA)
def SSA(Initial, StateChangeMat, FinalTime):
    # Inputs:
     #  Initial: initial conditins of size (StateNo x 1)
     #  StateChangeMat: State-change matrix of size (ReactNo, StateNo)
     #  FinalTime: the maximum time we want the process be run
    # Outputs:
     #  AllTimes: the dict. of all selected time levels
     #  AllStates: the dict. of all state values at corresponding time levels
    [m,n] = StateChangeMat.shape
    ReactNum = np.array(range(m))
    AllTimes = {}   # define a dict. for storing all time levels
    AllStates = {}  # define a dict. for storing all states at all time levels
    AllStates[0] = Initial
    AllTimes[0] = [0]
    k = 0; t = 0; State = Initial
    while True:
        w = PropensityFunc(State, m)     # propensities
```

```
        a = np.sum(w)
        tau = RandExp(a,1)                # WHEN the next reaction happens
        t = t + tau                       # update time
        if t > FinalTime:
            break
        which = RandDisct(ReactNum,w/a,1)             # WHICH reaction occurs
        State = State + StateChangeMat[which.item(),] # Uppdate the state
        k += 1
        AllTimes[k] = t
        AllStates[k] = State
    return AllTimes, AllStates
```

The above `SSA` function works for a general stochastic model provided that the propensity functions and state-change vectors are given. Below, we will specify these functions for the SIR model discussed earlier.

```
mu, bet, gam = 1e-4, 0.25, 0.05  # rates
Initial = [198,2,0]              # initial values
FinalTime = 120                  # final time
def PropensityFunc(State, ReactNo):
    S,I,R = State
    N = S + I + R;
    w = np.zeros(ReactNo)
    w[0] = mu * N              # birth (newborns)
    w[1] = bet/N * S * I       # infection
    w[2] = gam * I             # recovery
    w[3] = mu * S              # death of susceptible individuals
    w[4] = mu * I              # death of infected individuals
    w[5] = mu * R              # death of recovered individuals
    return w
StateChangeMat = np.array([
                    [+1,  0,  0],
                    [-1, +1,  0],
                    [ 0, -1, +1],
                    [-1,  0,  0],
                    [ 0, -1,  0],
                    [ 0,  0, -1]])
```

Finally, we call the `SSA` function to run multiple simulations, for instance $N = 10$, and

subsequently plot the results.

```
N = 10   # number of simulations
plt.figure(figsize = (6, 4))
for k in range(N):
    Time, States = SSA(Initial, StateChangeMat, FinalTime)
    n = len(Time)
    t = [Time[i][0] for i in range(n)]
    S = [States[i][0] for i in range(n)]
    I = [States[i][1] for i in range(n)]
    R = [States[i][2] for i in range(n)]
    plt.plot(t,S,linestyle = '-', color='blue')
    plt.plot(t,I,linestyle = '-', color='red')
    plt.plot(t,R,linestyle = '-', color='green')
plt.xlabel('Time');
plt.ylabel('Individuals');
plt.title('Stochastic solutions using SSA')
plt.legend(['$S$','$I$','$R$'],loc='center right')
plt.show()
```

The results are given in Figure 23 for 10 simulations of the above SSA algorithm. This figure represents the stochastic nature of the SIR model.

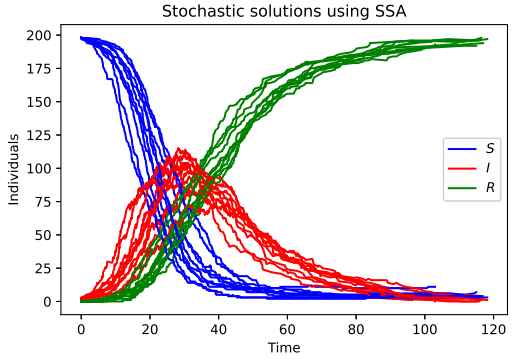


Figure 23: Stochastic simulation of the SIR model using the Gillespie algorithm: ten simulations

One can execute the code and obtain many simulations (stochastic processes) and finally compute the mean of the solutions as a Monte Carlo solution. In many cases the mean is close to the deterministic solution of the counterpart ODE model.

To speed up the SSA, Daniel Gillespie proposed another algorithm called *tau-leaping*[6], which replaces the variable waiting times between reactions with a fixed step size τ for each iteration.

[6]Daniel T. Gillespie, *Approximate accelerated stochastic simulation of chemically reacting systems*, The Journal of Chemical Physics. 115 (2001) 1716-1733.

Unlike the standard SSA, where only one reaction occurs per step, tau-leaping allows multiple reactions to occur in a single time step, and the state changes can be larger than just 1. For each reaction j, the state change is sampled from a Poisson distribution with parameter $\lambda = w_j\tau$, where w_j is the propensity of reaction j, and τ is the step length. Such Poisson sample represents the number of events expected in an interval of length τ with propensity w_j. This algorithm requires careful handling to avoid negative populations when some states are close to zero and state changes are large. Additionally, selecting an optimal step size τ is a key challenge because steps that are too large can lead to inaccuracies, while steps that are too small reduce the performance benefit of the algorithm.

Finally, we note that the Gillespie algorithm is also available in the `GillesPy2` library, which provides an object-oriented framework for building and simulating the mathematical model. The methods include the SSA, the tau-leaping, and some numerical ODE solvers. The library is optimized for performance, and is written in C++ and NumPy. For more details check this link[7].

7.3 Application to biochemical kinetics

In biochemical systems, a finite number of particles present in living cells and the inherent randomness associated with molecular interactions and reaction rates lead to a need for models that account for discrete and stochastic dynamics rather than continuous and deterministic ones.

Consider a model comprised of n species, denoted as $\{S_1, S_2, \ldots, S_n\}$, which interact through m chemical reactions, represented as $\{r_1, \ldots, r_m\}$. As before, the state of the system can be described by the vector $\boldsymbol{y}(t) = (y_1(t), \ldots, y_n(t))$, with the initial condition specified as $\boldsymbol{y}(0) = \boldsymbol{y}_0$. The interactions are modeled by reactions, propensity functions, and state-change vectors. We assume that each reaction r_ℓ is *elemental* and is either *unimolecular* or *bimolecular*. For simplicity, let us consider a system with three species with state variable $\boldsymbol{y} = (y_1, y_2, y_3)$. The following table illustrates various reaction cases, where c denotes a constant rate:

Reaction	State-change vector	Propensity function
$y_1 \xrightarrow{c} y_2$	$\boldsymbol{v} = [-1, 1, 0]$	$w = cy_1$
$y_1 \xrightarrow{c} y_2 + y_3$	$\boldsymbol{v} = [-1, 1, 1]$	$w = cy_1$
$y_1 \xrightarrow{c} y_1 + y_2$	$\boldsymbol{v} = [0, 1, 0]$	$w = cy_1$
$y_1 \xrightarrow{c} 2y_1$	$\boldsymbol{v} = [1, 0, 0]$	$w = cy_1$
$y_1 \xrightarrow{c} \emptyset$	$\boldsymbol{v} = [-1, 0, 0]$	$w = cy_1$
$y_1 + y_2 \xrightarrow{c} y_3$	$\boldsymbol{v} = [-1, -1, 1]$	$w = cy_1y_2$
$2y_1 \xrightarrow{c} y_2$	$\boldsymbol{v} = [-1, 1, 0]$	$w = cy_1(y_1 - 1)/2$
$2y_1 \xrightarrow{c} y_1$	$\boldsymbol{v} = [-1, 0, 0]$	$w = cy_1(y_1 - 1)/2$
$\emptyset \xrightarrow{c} y_1$	$\boldsymbol{v} = [1, 0, 0]$	$w =?$

[7] `https://gillespy2.readthedocs.io/en/latest/`

The propensity functions reflect the concentration of the reactants and emphasize how molecular interactions dictate the rates of reaction.

As an example consider the *Michaelis-Menten system* which is a standard model for enzyme-catalyzed reactions. In this model, we consider a substrate S, enzyme E, the enzyme-substrate complex C, and the product P. The model below shows the series of reactions occurring in this system.

$$S + E \underset{c_2}{\overset{c_1}{\rightleftharpoons}} C \xrightarrow{c_3} P + E$$

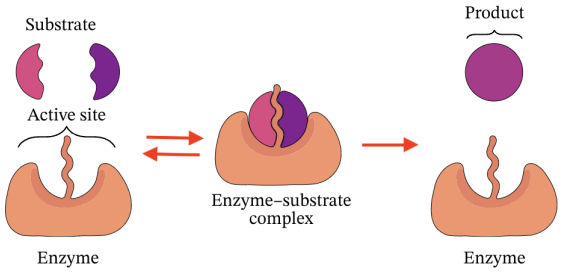


Figure 24: The Michaelis-Menten reactions (image from `www.nagwa.com/en/`)

If we denote the state vector by $\boldsymbol{y} = (S, E, C, P)$ then the reactions can be summarized as follows:

$$\begin{aligned}
r_1 &: \quad S + E \xrightarrow{c_1} C, \quad \boldsymbol{v}_1 = [-1, -1, +1, 0], \quad w_1 = c_1 SE = c_1 y_1 y_2 \\
r_2 &: \quad S + E \xleftarrow{c_2} C, \quad \boldsymbol{v}_2 = [+1, +1, -1, 0], \quad w_2 = c_2 C = c_2 y_3 \\
r_3 &: \quad C \xrightarrow{c_3} P + E, \quad \boldsymbol{v}_3 = [0, +1, -1, +1], \quad w_3 = c_3 C = c_3 y_3
\end{aligned}$$

The first reaction involves the formation of the enzyme-substrate complex. The second reaction is the reverse process, where the complex dissociates back into the substrate and enzyme. The third reaction describes the conversion of the complex into product and enzyme. For given constant rates c_j and initial numbers of proteins, we can simply solve this model using the Gillespie algorithm.

7.4 Lotka-Volterra models

Other important examples are the *Lotka-Volterra models* which are used to describe predator-prey interactions. Predator and prey animals, such as hawks and mice or foxes and rabbits, interact in an ecosystem where predator eat prey. If the prey population is large, food becomes easily available for the predator and the population grows. This leads to a decrease in the prey population, which, in turn, leads to reduced access to food, resulting in a decrease in the predator population. This means that more prey survives, and so on. This *process* continues indefinitely.

As an example of a simple Lotka-Volterra model assume that R represents the number of prey, e.g. rabbits, and F represents the number of predators, e.g. foxes. To analyse the predator-prey system, we consider the following set of reactions:

$$R \xrightarrow{\alpha} 2R$$
$$R + F \xrightarrow{\beta} 2F$$
$$F \xrightarrow{\gamma} \emptyset$$

The first reaction incorporates prey reproduction into our model, with α denoting the reproduction rate of each prey. The second reaction incorporates predator reproduction per prey, with β denoting the rate of interactions between prey and predator animals. In other words, the predator consumes prey and reproduces at a rate of β. The third and final reaction incorporates predator mortality, with γ denoting the rate at which predators are removed from the ecosystem. Now, we can apply the Gillespie algorithm to solve this simple model provided that the initial populations F_0 and R_0 and the model rates are given. This task is left as an exercise to the reader.

8 Markov chain Monte Carlo (MCMC)

In preceding sections we have typically generated iid random variables directly from the density of interest f. In this section the *Markov chain Monte Carlo (MCMC)* is introduced as a powerful tool to *approximately* generate samples from an *arbitrary* distribution[8]. The main idea behind the MCMC algorithms is to simulate a Markov chain such that its stationary distribution approximately coincides with the desired distribution f. One of the MCMC algorithms is the *Metropolis-Hastings* algorithm which is discussed in detail here. We follow [**Rubinnstein-Kroese:2017**] in this section.

8.1 Metropolis-Hastings algorithm

Given a target density f, we want to generate a Markov chain $\{X_t : t = 0, 1, \ldots\}$ with stationary distribution f. For simplicity, we start with a general discrete distribution. Assume that we want to generate a random variable X which takes its values in sample space

$$\mathcal{S} = \{1, 2, \ldots, m\}$$

with target distribution

$$\{\pi_1, \pi_2, \ldots, \pi_m\}.$$

[8]The MCMC method was motivated by the pioneer work of Metropolis, et. al. in 1953:
M. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *Equations of state calculations by fast computing machines,* Journal of Chemical Physics, 21(1953) 1087-1092.
After that, many modifications were done on the original MCMC algorithm, notably the Hastings algorithm in 1970:
W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, 57(1970) 92-109.
An alternative methodology then proposed by German and German in 1984 which is known as Gibbs sampler:
S. Geman and D. Geman, *Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images*, IEEE Transactions on PAMI, 6(1984) 721-741.

In the Metropolis-Hastings algorithm a Markov chain $\{X_t : t = 0, 1, \ldots\}$ on $\mathcal{S}$ is simulated based on a primary transient matrix $Q = (q_{ij})$. This matrix is used to approximate the actual transient matrix of the chain. The algorithm contains the following steps:

(1) [*Variable generation*] Given $X_t = i$, generate a random variable Y such that $\mathbb{P}(Y = j) = q_{ij}$ for all $j \in \mathcal{S}$. On the other words, generate Y from the i-th row of Q.

(2) [*Accept or reject*] If $Y = j$ then accept Y with probability α_{ij} and let $X_{t+1} = Y$, otherwise reject Y and let $X_{t+1} = X_t$, where

$$\alpha_{ij} = \min\left\{\frac{\pi_j}{\pi_i}\frac{q_{ji}}{q_{ij}}, 1\right\}. \tag{8.1}$$

Since X_{t+1} is obtained from X_t only, the chain is Markov. Besides, the transient matrix $P = (p_{ij})$ of the chain is

$$p_{ij} = \begin{cases} q_{ij}\alpha_{ij}, & i \neq j \\ 1 - \sum_{k \neq i} q_{ik}\alpha_{ik}, & i = j \end{cases}, \tag{8.2}$$

because, for $i \neq j$ we can write

$$\begin{aligned} p_{ij} &= \mathbb{P}(X_{t+1} = j \,|\, X_t = i) \\ &= \mathbb{P}(Y = j, Y \text{ is accepted} \,|\, X_t = i) \\ &= \mathbb{P}(Y = j \,|\, X_t = i) \times \mathbb{P}(Y \text{ is accepted} \,|\, Y = j, X_t = i) \\ &= q_{ij} \times \alpha_{ij}. \end{aligned}$$

In the third equality we have used the identity $\mathbb{P}(A \cap B|C) = \mathbb{P}(A|C)\mathbb{P}(B|A \cap C)$. The case $i = j$ in (8.2) follows from the fact that each row of P sums up to unity. Using (8.2) and the definition of α_{ij} in (8.1) we have

$$\pi_i p_{ij} = \pi_j p_{ji}, \quad i, j \in \mathcal{S} \tag{8.3}$$

which is the *detailed balance equation* for the Markov chain. The proof is as follows. First we have

$$\pi_i p_{ij} = \pi_i q_{ij}\alpha_{ij} = \pi_i q_{ij}\frac{\pi_j}{\pi_i}\frac{q_{ji}}{q_{ij}} = \pi_j q_{ji}$$

provided that $\frac{\pi_j}{\pi_i}\frac{q_{ji}}{q_{ij}} \leqslant 1$. On the other hand

$$\pi_j p_{ji} = \pi_j q_{ji}\alpha_{ji} = \pi_j q_{ji}$$

because $\alpha_{ji} = 1$ from the fact that $\frac{\pi_i}{\pi_j}\frac{q_{ij}}{q_{ji}} \geqslant 1$.

Taking summation on j from both sides of equation (8.3) gives $\pi_i = \sum_j \pi_j p_{ji}$ for $i \in \mathcal{S}$, or $\boldsymbol{\pi} = \boldsymbol{\pi}P$, which means that the chain has stationary probability $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_m]$. This stationary distribution is also a limiting distribution if the Markov chain is irreducible and aperiodic.

An important advantage of the above Metropolis-Hastings algorithm is that the target distribution $\boldsymbol{\pi}$ needs to be only known up to a normalization constant C, because in the definition of α_{ij} (the only place $\{\pi_j\}$ is used) any constant C will be cancelled in quotient $\frac{\pi_j}{\pi_i}$.

The above algorithm can be generalized to generate samples from an arbitrary multi-

dimensional density $f(x)$ instead of $\{\pi_j\}$. From here on x, y, X and Y with or without subscripts are d-dimensional variables. In this generalization, the non-negative transient kernel $q(x, y)$ will be used instead of the transient matrix Q. Since the transient kernel is a conditional pdf, we can also write $q(y|x)$ instead of $q(x, y)$. This kernel is usually called the *proposal* function, and plays a role similar to proposal distribution g in the acceptance-rejection method of section 3.2.

The Metropolis-Hastings algorithm starts with an initial state X_0 and a target pdf $f(x)$, a proposal function $q(x, y)$ and the number of required samples N as inputs and generates a Markov chain $X_1, X_2, \ldots, X_N$ approximately distributed according to $f(x)$. The algorithm is given below.

Algorithm 6 Metropolic-Hastings Algorithm

Require: Target distribution f, Proposal distribution q, Initial state X_0, Number of samples N

for $t = 1, 2, \ldots, N$ **do**

1. Given X_t, generate $Y \sim q(X_t, y)$
2. Set $\alpha = \min\left\{\dfrac{f(Y)}{f(X_t)}\dfrac{q(Y, X_t)}{q(X_t, Y)}, 1\right\}$
3. Generate $U \sim \mathcal{U}(0, 1)$
4. If $U \leqslant \alpha$ accept Y and set $X_{t+1} = Y$, otherwise reject Y and set $X_{t+1} = X_t$.

end for

Ensure: The sequence $X_1, X_2, \ldots, X_N$

Starting by X_0, we continue this process until X_N is generated. The sequence $X_1, \ldots, X_N$ is a set of dependent random variables and X_t for large t is approximately distributed according to $f(x)$.

The original Metropolis algorithm uses a proposal function q with symmetrical property $q(x, y) = q(y, x)$, while the modified version by Hastings allows the nonsymmetric kernels as well. With a symmetric q, the probability α reduces to

$$\alpha = \min\left\{\frac{f(Y)}{f(X_t)}, 1\right\}. \tag{8.4}$$

This does not mean that q is ruled out because Y is still generated from q.

The simplest proposal function is to take

$$q(x, y) = g(y)$$

for some pdf $g(y)$. Using this proposal function, in step (1) of the Metropolis-Hastings algorithm, Y is generated from $g(y)$ independent of the current variable X_t. The acceptance probability α then is

$$\alpha = \min\left\{\frac{f(Y)}{f(X_t)}\frac{g(X_t)}{g(Y)}, 1\right\}$$

which depends on X_t. Thus the chain still produces dependent samples.

In a *random walk sampler* the current state Y for a given state x is given by $Y = x + Z$ where Z is generated from a radially symmetric distribution such as $\mathcal{N}(0, \Sigma)$. For this case Y

is indeed generated from $Y \sim \mathcal{N}(x, \Sigma)$, i.e.,

$$q(x, y) = (2\pi)^{-d/2} \exp\left(-\tfrac{1}{2}(x-y)^T \Sigma (x-y)\right).$$

Since the proposal function is symmetric the acceptance probability is reduced to (8.4). A Phyton code for MCMC with multidimensional normal random walk sampler is given here. Inputs are the desired probability density function `pdf` we aim to sample from, the initial state `X0`, the covariance matrix of the proposal function q and the number of samples we ask for. The output is a Markov chain `X`. The `RandMultiNormal` function is given in Section A.9.

```
def McMcRandWalkGen(pdf, X0, SigmaWalk, N):
    dim = np.size(X0)
    X = np.zeros([dim,N])
    X[:,0] = X0
    for t in range(N-1):
        Z = RandMultiNormal(np.zeros(dim),SigmaWalk,1).T
        Y = X[:,t] + Z
        Xt = np.array([X[:,t]])
        alpha = min(pdf(Y)/pdf(Xt),1)
        U = np.random.rand()
        if U <= alpha:
            X[:,t+1] = Y
        else:
            X[:,t+1] = X[:,t]
    return X
```

Example 8.1 (Rubinnstein-Kroese:2017)**.** Consider a random vector $X = [X_1, X_2]$ with the following bivariate pdf

$$f(x, y) = c \exp\Big(-(x^2y^2 + x^2 + y^2 - 8x - 8y)/2\Big) \tag{8.5}$$

where $c \doteq 1/20216.335877$ is the normalization constant.

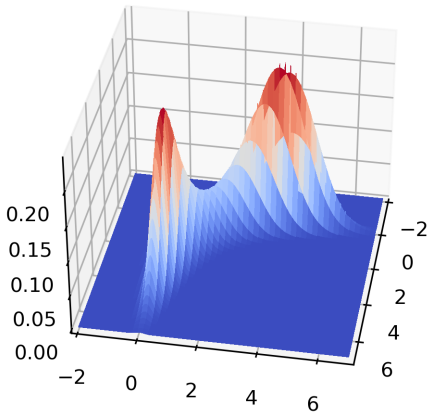


Figure 25: Surface plot of bivariate density function $f(x, y)$.

The surface graph of this density is shown in Figure 25. We want to generate samples $\{X_t = (X_{t1}, X_{t2}) : t = 1, 2, \ldots, N\}$ from $f(x, y)$ using the Metropolis-Hastings algorithm with the random walk sampler as a transition kernel. We assume that $\Sigma = \text{diag}[\sigma^2, \sigma^2]$ where the moderate value $\sigma = 2$ is chosen in our experiment. We run the following script to produce $N = 10^4$ samples from f. The contour plot of f and the samples are displayed in Figure 26. We discarded an initial "burn-in" period (say the first 1000 samples) to ensure the chain has reached a stable distribution. We observe that the correct region is sampled.

```
def f(xy):          # Define the distribution
    x,y = xy[0];  c = 1/20216.335877
    return c*np.exp(-(x**2*y**2+x**2+y**2-8*x-8*y)/2)
Sigma = 2*np.eye(2)
N = 10**4           # number of MH samples
X0 = [0,0]          # initial guess
X = McMcRandWalkGen(f, X0, Sigma, N)  # Call the MCMC function
#      plot the results
xeval = np.linspace(-1, 7, 1000)
[x,y] = np.meshgrid(xeval,xeval)
feval = f([[x,y]])
plt.figure(figsize = (5, 5))
plt.contour(x,y,feval)
plt.figure(figsize = (5, 5))
plt.plot(X[0,1000:], X[1,1000:], color = 'red',
         marker = 'o', markersize = 2, linestyle = '')
```

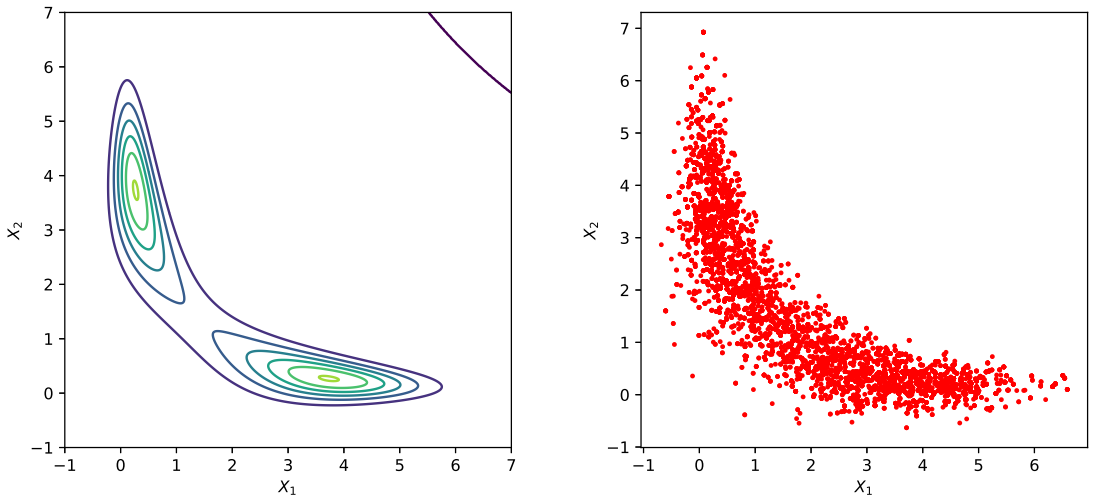


Figure 26: The contour plot of bivariate distribution f (left) and the samples using the Metropolis-Hastings algorithm (right). The first 1000 samples are discarded.

The histogram of variable X_1 is also shown in Figure 27. It is close to the true marginal pdf. We use a numerical integration to compute the exact marginal distribution via formula

$$f_{X_1}(x) = \int_{-\infty}^{\infty} f(x, y)dy.$$

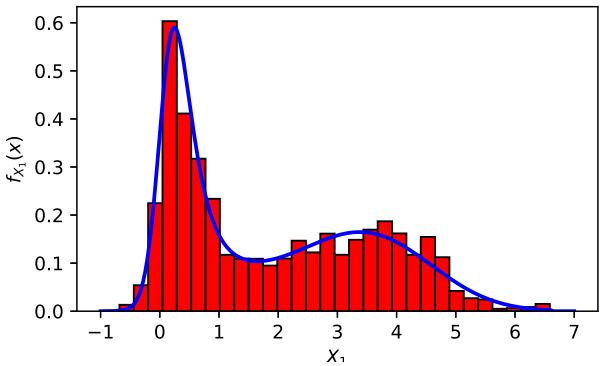


Figure 27: The histogram of the marginal variable X_1, and the true corresponding marginal pdf (the blue curve).

Now, assume that we want to estimate $\mathbb{E}_f(X_1)$ using the Monte Carlo method. It is enough to compute the mean of marginal samples $\{X_{t1}\}$.

```
MeanVals = np.zeros(4)
for k in range(4):
    N = 10**(k+3)
    X0 = [0,0]
    X = McMcRandWalkGen(f, X0, SigmaWalk, N)
    MeanVals[k] = np.mean(X[0,1000:])
print("MCMC estimates = ", np.round(MeanVals,4))
```

The output of a run is

```
MCMC estimates =  [1.6999 1.8169 1.8985 1.8584]
```

which actually four estimations for $\mathbb{E}_f(X_1)$ with $N = 10^3, 10^4, 10^5$ and 10^6. The exact value is $\mathbb{E}_f(X_1) \doteq 1.85997$.

8.2 MCMC Bayesian parameter estimation

One application of the Metropolis-Hastings algorithm arises in Bayesian inference for parameter estimation. In Bayesian inference, we aim to estimate parameters $\theta = (\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(d)})$ of a model by calculating the *posterior distribution* $p(\theta|\text{data})$, which reflects our updated beliefs about the parameters after observing the data. The posterior distribution is typically

computed as

$$p(\theta|\text{data}) = p(\text{data}|\theta) \cdot p(\theta)$$

where $p(\text{data}|\theta)$ is the *likelihood* of observing the data given the parameters, and $p(\theta)$ is the *prior distribution*, which encodes our initial beliefs about θ before seeing the data.

However, calculating $p(\theta|\text{data})$ explicitly can be difficult, especially if the likelihood is complex or the dimensionality of θ is high. This is where the Metropolis-Hastings algorithm becomes useful. We assume that the proposal distribution q is given and $f(\theta) = p(\text{data}|\theta) \cdot p(\theta)$. We start with an initial guess $\theta_0 = (\theta_0^{(1)}, \ldots, \theta_0^{(d)})$ for the parameters and we use Algorithm 6 to generate a chain of parameter samples. After a sufficient number of iterations the generated samples $\{\theta_1, \theta_2, \ldots, \theta_N\}$ approximate the target posterior distribution $p(\theta|\text{data})$. We can now use the Metropolis-Hastings samples to estimate the *posterior* mean, variance, and confidence intervals for each parameter.

Example 8.2 (Estimating Patient Recovery Rate)**.** Suppose we observe the recovery times of 10 patients after treatment, given in days as

$$\text{data} = \{5, 8, 12, 7, 9, 10, 3, 6, 8, 11\}.$$

Our goal is to estimate the posterior distribution of the recovery rate parameter θ, which represents the average rate at which patients recover. We assume that recovery times follow an exponential distribution

$$p(x|\theta) = \theta e^{-\theta x}.$$

The aim is to estimate the parameter θ by approximating its (posterior) distribution, i.e., generating samples from its distribution that is unknown to us. But we assume that θ has a prior distribution. In this example we will use the *Gamma* distribution as a prior for θ, i.e.,

$$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \theta^{\alpha-1} e^{-\beta\theta}$$

where $\alpha = 2$ and $\beta = 1$, which encodes our prior belief that θ is likely around 2 because the mean of the Gamma distribution is α/β.

For the given independent recovery times x_j (our data), the likelihood of observing the data given θ is

$$p(\text{data}|\theta) = \prod_{j=1}^{10} \theta e^{-\theta x_j}.$$

Note that, in the likelihood θ is the parameter of the distribution while in the prior θ is the variable of the distribution.

We start with an initial guess $\theta_0 = 1$ and use the random walk sampler (normal distribution $\mathcal{N}(\theta_t, \sigma^2)$) with a small variance $\sigma^2 = 0.25$ at each time step t to generate the new parameter θ_{t+1} using the Metropolis-Hastings algorithm. The code is given below. Note that to define the distribution $f(\theta) = p(\text{data}|\theta) \cdot p(\theta)$ we use the logarithm of likelihood and prior

distributions to compute $\log(f(\theta))$ and finally return $f(\theta) = \exp(\log(f(\theta)))$.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma, expon   # Gamma and Exponential distributions

# Given data
data = [5,8,12,7,9,10,3,6,8,11]
# Define MCMC distribution = likelihood * priors
def mcmc_pdf(theta):
    log_likelihood = np.sum(expon.logpdf(data, scale = 1/theta))
    alpha = 2; beta = 1;
    log_prior = gamma.logpdf(theta, alpha, scale = 1/beta)
    return np.exp(log_likelihood+log_prior)

Sigma = 0.5   # std value for random walk sampler (transient kernel)
theta0 = .5   # initial guess
N = 10**4     # length of MCMC chain
BurnIn = 1000 # burn-in period for MH

Theta = McMcRandWalkGen(mcmc_pdf, theta0, Sigma, N)  # Call MCMC algorithm
Theta = Theta[:,BurnIn:]          # Discard an initial burn-in period
MeanTheta = np.mean(Theta)

err = 1.96*np.std(Theta)/np.sqrt(np.size(Theta))
print("Posterior mean of parameter is ${0} +/- {1} with 95% of probability"
      .format(np.round(MeanTheta,4), np.round(err,4)))

plt.hist(Theta[0,:], bins=30, density=True, alpha=0.6, color='red')
plt.axvline(MeanTheta, color='b', linestyle='dashed', linewidth=1.5,
                      label=f"Mean recovery rate: {MeanTheta:.4f}")
plt.title("Distribution of recovery rate parameter")
plt.xlabel("Recovery rate")
plt.ylabel("Density")
plt.legend()
plt.show()
```

The histogram of θ samples is plotted and together with its estimated mean value are shown in Figure 28.

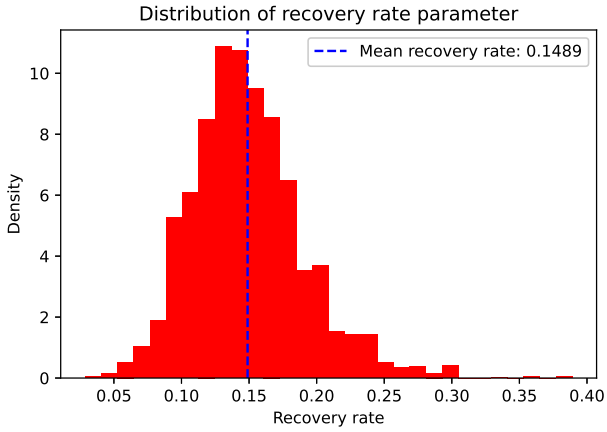


Figure 28: Posterior distribution of parameter θ sampled by the Metropolis-Hastings algorithm, and its estimated mean value.

We also calculated the standard deviation of θ values and a confidence interval with 95% probability to get an estimate for the error of the recovery rate. The output is

```
Posterior mean of parameter is $0.1489 +/- 0.0009 with 95% of probability
```

Example 8.3 (Estimating the probability of a large portfolio loss)**.** A financial analyst wants to estimate the likelihood that an investment portfolio valued at $\$1,000,000$ will lose more than $\$100,000$ over a 6-month period. This probability, often referred to as the *Value at Risk (VaR)* at a certain confidence level, is crucial for managing risk and setting capital reserves.

We assume the portfolio value S_t follows a geometric Brownian motion

$$\mathrm{d}S_t = \mu S_t \,\mathrm{d}t + \sigma S_t \,\mathrm{d}W_t$$

which is commonly used to model stock prices, as discussed in Example 6.4. Here μ is the drift rate (expected return rate of the portfolio), σ is the volatility of returns, and W_t is a Wiener process. We learnt in Section 6.3 how to generate a sample path S_t using the Euler-Maruyama method.

The goal here is to estimate the posterior distributions of parameters $\theta = (\mu, \sigma)$ given historical data on portfolio returns. From these, then we can simulate future values of S_t and estimate the probability of a loss greater than $\$100,000$ over the next six months.

The analyst has monthly return rate data for the portfolio in the past five years, which we denote by

$$\text{data} = \{r_1, r_2, \ldots, r_{60}\}$$

where each r_j is the observed monthly return rate. Assuming that monthly returns are normally distributed, we define the likelihood of observing a monthly return r_j given μ and

σ as

$$p(r_j|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_j-\mu)^2}{2\sigma^2}\right).$$

For the entire dataset, the likelihood is

$$p(\text{data}|\mu,\sigma) = \prod_{j=1}^{60} p(r_j|\mu,\sigma).$$

It is remained to specify prior distributions for μ and σ. For μ, a normal distribution centered at the historical average return, say $\tilde{\mu}$, with standard deviation $\tilde{\sigma}$ is used, i.e.,

$$p_1(\mu) \sim \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2).$$

For σ, an Inverse-Gamma distribution

$$p_2(\sigma^2) \sim \mathcal{I}nv\mathcal{G}am(\alpha,\beta)$$

is used. The Inverse-Gamma distribution is commonly used for variance and precision parameters in Bayesian statistics, and has the distribution

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)}(1/x)^{\alpha+1}\exp(-\beta/x), \quad x>0, \quad \alpha,\beta>0.$$

This distribution has parameters α and β that can be chosen based on prior knowledge of the portfolio's volatility. Note that for $\alpha > 1$ the mean of the Inverse-Gamma distribution is $\beta/(\alpha-1)$. The joint prior distribution is defined as

$$p(\mu,\sigma^2) = p_1(\mu)\cdot p_2(\sigma^2).$$

To apply the Metropolis-Hastings algorithm we start with initial guesses for μ and σ, say $\theta_0 = (\mu_0,\sigma_0)$, and use a random walk sampler with covariance matrix

$$\Sigma = \begin{bmatrix} \tau_\mu^2 & 0 \\ 0 & \tau_\sigma^2 \end{bmatrix}$$

to propose a new parameter vector $\theta_{t+1} = (\mu_{t+1},\sigma_{t+1})$ at each step $t = 0,1,\ldots$ of the algorithm. Here τ_μ and τ_σ are small tuning parameters to control the step size of the sampler. We run the Metropolis-Hastings algorithm for many iterations (e.g., $N_{\text{MH}} = 10^4$), and discard the first few thousand samples for burn-in.

Finally, to estimate the VaR, we do the following steps:

1. (Simulate future portfolio values): Using the posterior samples of μ and σ, simulate the portfolio value after 6 months under the geometric Brownian motion model. For each sampled (μ,σ) pair, generate N_{MC} paths using function `DiffusionProcessGen` for drift coefficient $a(t,S_t) = \mu S_t$ and diffusion coefficient $b(t,S_t) = \sigma S_t$, and calculate the final portfolio values S_T for $T = 0.5$ a year (6 months). Finally, take a mean to have one estimate for the portfolio value for each pair (μ,σ).

2. (Calculate loss probability): Compute the proportion of paths where the portfolio value is below $\$900,000$ (a $\$100,000$ loss from the starting value). This proportion estimates the probability of a large loss in the period.

The Python code is given below where we use the input values as followings: The initial

portfolio value is $S_0 = 1,000,000$, the loss threshold is $900,000$ (a \$100,000 loss), the time frame is 6 months, the historical mean return is $\tilde{\mu} = 0.05$ (5% per month), the historical volatility is $\tilde{\sigma} = 0.1$, parameters for Inverse-Gamma distribution are $\alpha = 2$ and $\beta = 0.0004$, the proposal variances for μ and σ are $\tau_\mu = 0.001$ and $\tau_\sigma = 0.001$, the initial guess is $\theta_0 = (\mu_0, \sigma_0) = (\tilde{\mu}, \tilde{\sigma})$, the number of MCMC iterations is $N_{\text{MH}} = 10^4$, and the number of Monte Carlo paths per posterior sample is $N_{\text{MC}} = 10^3$. The data values r_j are given in the code.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import invgamma, norm  # import Inv-Gamma and normal dists.

# Parameters
S0 = 1000000               # Initial portfolio value
LossThreshold = 900000     # Loss threshold (100,000 loss)
FinalTime = 0.5            # Time horizon in years (6 months)
N_MH = 10000               # Number of Metropolis-Hastings iterations
N_MC = 1000                # Monte Carlo paths for each posterior sample
BurnIn = 2000              # Burn-in period for MH

# Prior parameters
mu_prior = 0.05            # Historical mean return
sigma_prior = 0.1          # Historical volatility
alpha_prior, beta_prior = 2, 0.0004

# Metropolis-Hastings proposal variances
tau_mu = 0.001
tau_sigma = 0.001

# Given data for sixty months
data = np.array(
  [0.07,  0.13,  0.10,  0.17,  0.11,  0.03,  0.15,  0.09,  0.12,  0.12,
  -0.06,  0.07,  0.09, -0.01,  0.08,  0.08,  0.07,  0.19,  0.09,  0.12,
   0.03,  0.16, -0.02,  0.2 ,  0.14,  0.05,  0.08,  0.06,  0.10, -0.07,
  -0.01, -0.07, -0.05,  0.21, -0.05,  0.02, -0.02,  0.15,  0.08,  0.02,
  -0.03,  0.01,  0.08,  0.13,   0.16, -0.03, -0.13, 0.14,  0.11,  0.12,
  -0.01, -0.07,  0.16,  0.27, -0.06,   0.01,  0.01, 0.01,  0.01,  0.16])
```

```
# Define MCMC distribution = likelihood * priors
def mcmc_pdf(theta):
    mu, sigma = theta[0]
    log_likelihood = np.sum(norm.logpdf(data, loc=mu, scale=sigma))
    log_prior_mu = norm.logpdf(mu, loc=mu_prior, scale=sigma_prior)
    log_prior_sigma = invgamma.logpdf(sigma**2, a=alpha_prior, scale=beta_prior)
    return np.exp(log_likelihood + log_prior_mu + log_prior_sigma)

# Call the Matropolis-Hastings algorithm
theta0 = [mu_prior, sigma_prior]   # initial guess theta0 = [mu0,sigma0]
SigmaWalk = np.diag([tau_mu,tau_sigma])     # covariance of the sampler
Theta = McMcRandWalkGen(mcmc_pdf, theta0, SigmaWalk, N_MH)

Theta = Theta[:,BurnIn:]   # Discard burn-in samples

# Monte Carlo simulation for each posterior sample (mu, sigma)
def drift(t,x,*args):      # drift
    mu = args[0]
    return mu*x
def diffusion(t,x,*args):  # diffusion
    sigma = args[1]
    return sigma*x
TimeStep = 0.01         # Time step in Euler-Maruyama method
LossPr = []             # Loss probability vector
ST = np.empty(N_MC)     # Stock value vector at final time
for mu, sigma in zip(Theta[0,:], Theta[1,:]):  # loop over all samples
    for j in range(N_MC):                      # Monte Calro loop
        ST[j] = DiffusionProcessGen(drift, diffusion, [0,FinalTime],
                                    TimeStep, S0, mu, sigma)[-1]
    LossPr.append(np.mean(ST < LossThreshold))

# Calculate the mean loss probability
MeanLossPr = np.mean(LossPr)

# Results
print(f"Estimated Probability of a $100,000 Loss: {MeanLossPr:.4f}")

# Plotting the results
```

```
plt.figure(figsize=(6, 4))
plt.hist(LossPr, bins=30, density=True, alpha=0.6, color='red')
plt.axvline(MeanLossPr, color='b', linestyle='dashed', linewidth=1.5,
                        label=f"Mean Loss Probability: {MeanLossPr:.4f}")
plt.title('Distribution of Loss Probabilities from MC Simulations')
plt.xlabel('Loss Probability')
plt.ylabel('Density')
plt.legend()
plt.show()
```

We computed the mean probability of loss from all simulations and visualize the distribution of the estimated loss probabilities by drawing the histogram of the loss probability values. See Figure 29. The estimated probability of a $\$100,000$ loss is about 1%.

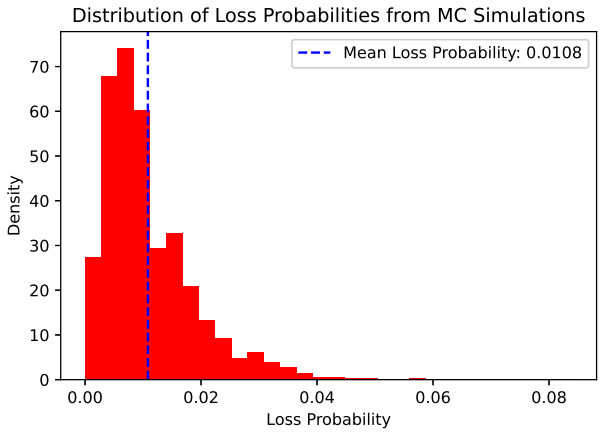


Figure 29: The distribution of the estimated loss probabilities and its estimated mean.

The main advantage of MCMC is that it can be used to generate random samples from any target distribution, regardless of its dimensionality and complexity. Main disadvantages are (1) The resulting samples are often highly correlated, (2) Sometimes N should be large so that the Markov chain settles down to its steady state, (3) the estimates obtained by MCMC often have greater variances than those obtained from iid samplings. During the past years, modified versions and more efficient MCMC algorithms have been developed to overcome such disadvantages. See for example [**Robert-Casella:2004**].

9 Exercises

This section includes a number of exercises designed to deepen your understanding of the lecture material. Some of the exercises are adapted from the references listed in the bibliography.

Exercise 9.1. The *Weibull distribution* is named after Swedish mathematician Waloddi Weibull, who described it in detail in 1939. The pdf (probability density function) of this distribution is defined by two parameters λ (scale parameter) and α (shape parameter), and is given by

$$f(x) = \frac{\alpha}{\lambda}\left(\frac{x}{\lambda}\right)^{\alpha-1} e^{-(x/\lambda)^{\alpha}}, \quad x \geq 0.$$

Demonstrate how the Inverse Transform Method (ITM) can be used to generate random numbers from a Weibull distribution. Write down all the steps. Then design a Monte Carlo algorithm for estimating the value of integral

$$I = \int_0^{\infty} \sqrt{1+x^2} f(x)\, dx$$

where $f(x)$ is a Weibull pdf with parameters $\alpha = 2.5$ and $\lambda = 1$. Be sure to not only return the estimation of I but also an estimate of the error.

Exercise 9.2. Consider the integral

$$I = \int_0^{\pi} (x + \sqrt{x}) \sin x \, \mathrm{d}x.$$

Let us consider I as an expectation value of a random variable $g(X)$ where X has a sine-distribution with pdf

$$f(x) = \begin{cases} \frac{1}{2}\sin(x), & 0 \leq x \leq \pi \\ 0, & otherwise \end{cases}.$$

We write $X \sim \sin$. Given uniformly distributed random numbers $U \sim \mathcal{U}(0,1)$, write with details that how you convert these into sine distributed random numbers. Then write I as an expectation value using such a random variable $X \sim \sin$ and write a pseudocode that estimates the integral using a Monte Carlo method with a given number of N samples. Be sure to not only return estimation of I but also an estimate of the error.

Exercise 9.3. Let $X_1, \ldots, X_n$ be iid random variables with cdf F. Assume that

$$X_{(1)} := \min\{X_1, \ldots, X_n\}, \quad X_{(n)} := \max\{X_1, \ldots, X_n\}.$$

First prove that the cdf of $X_{(n)}$ is $F_n(x) = [F(x)]^n$ and the cdf of $X_{(1)}$ is $F_1(x) = 1 - [1 - F(x)]^n$. Then show that

$$X_{(n)} = F^{-1}(U^{1/n}), \quad X_{(1)} = F^{-1}(1 - U^{1/n})$$

where $U \sim \mathcal{U}(0,1)$. Random variables $X_{(1)}$ and $X_{(n)}$ are called *ordered statistics*. Show how inverse transform method can be used to sample from ordered statistics.

Exercise 9.4. How the inverse transform method can be applied to generate from Beta distributions $\mathcal{B}eta(\alpha, 1)$ and $\mathcal{B}eta(1, \beta)$. Derive the formulation and implement the Python code.

Exercise 9.5. Explain how we can generate a random variable X from the semicircular pdf

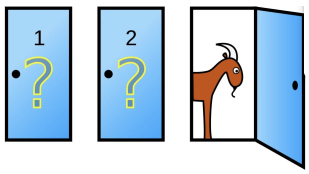
$$f(x) = \frac{2}{\pi R^2}\sqrt{R^2 - x^2}, \quad x \in [-R, R]$$

using the acceptance-rejection algorithm. Then use the `RandAcceptReject` function for illustration.

Exercise 9.6. Write a Python code for drawing normal samples using the acceptance-rejection method with exponential distribution as a proposal distribution. Refer to discussions at the end of subsection 3.2. Plot histograms for different number of samples.

Exercise 9.7. Develop an algorithm for sampling from $\mathcal{B}in(p, n)$ for large values of n using the fact that the distribution of a binomial variable $X \sim \mathcal{B}in(p, n)$ is close to that of $Y \sim \mathcal{N}(np - 1/2, np(1 - np))$ for a large n.

Exercise 9.8 (Monty Hall Problem)**.** You are on a game show, being asked to choose between three doors. One door has a car, and the other two have goats. The host, Monty Hall, opens one of the other doors, which he knows has a goat behind it. Monty then asks whether you would like to switch your choice of door to the other remaining door. Do you choose to switch or not to switch? Solve it with Monte Carlo method.
Answer: If you switch you win the car with 2/3 probability. Image source: Wikipedia.



Exercise 9.9. For the normal-Cauchy Bayes estimator

$$\delta(t) = \int_{-\infty}^{\infty} \frac{x}{1+x^2} \mathrm{e}^{-(x-t)^2/2} \mathrm{d}x \Big/ \int_{-\infty}^{\infty} \frac{1}{1+x^2} \mathrm{e}^{-(x-t)^2/2} \mathrm{d}x$$

use the Monte Carlo integration based on normal simulations to estimate $\delta(t)$ for $t = 0, 2, 4$. Monitor the convergence with the standard error of the estimate. Determine the minimum value N to obtain three digits of accuracy with 0.95% probability.

Exercise 9.10. Write down the proof of Theorem 5.2.

Exercise 9.11. Suppose that the morning weather of a city in a time period can be only sunny or cloudy, and the weather conditions on successive mornings form a Markov chain with transition matrix

$$\begin{array}{c} \\ \text{sunny} \\ \text{cloudy} \end{array} \begin{array}{c} \begin{array}{cc} \text{sunny} & \text{cloudy} \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} \end{array} .$$

1. If it is cloudy on a given day, what is the probability that it will also be cloudy the next day?
2. If it is sunny on a given day, what is the probability that it will be sunny on the next two days?
3. If it is cloudy on a given day, what is the probability that it will be sunny on at least one of the next three days?
4. If it is sunny on a certain Wednesday, what is the probability that it will be sunny on the following Saturday?
5. If it is cloudy on a certain Wednesday, what is the probability that it will be sunny on the following Saturday?
6. If it is sunny on a certain Wednesday, what is the probability that it will be sunny on both the following Saturday and Sunday?
7. If it is cloudy on a certain Wednesday, what is the probability that it will be sunny on both the following Saturday and Sunday?
8. Suppose that the probability that it will be sunny on a certain Wednesday is 0.2 and the probability that it will be cloudy is 0.8. Determine the probability that it will be cloudy on the next day, Thursday.
9. With assumptions of item 8, determine the probability that it will be cloudy on Friday.

Exercise 9.12. Suppose that a Markov chain has state space $\mathcal{S}$ with four states $\{1, 2, 3, 4\}$ and transition matrix

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0.25 & 0.25 & 0.00 & 0.50 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.50 & 0.00 & 0.50 & 0.00 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \end{array}$$

If the chain is in state 3 at a given time t, what is the probability that it will be in state 2

at time $t+2$? If the chain is in state 1 at a given time t, what is the probability that it will be in state 3 at time $t+3$?

Exercise 9.13. We want to use a finite Markov chain to model the probability of customers making purchases based on their past behavior. Assume that customers can be in one of the following four states:

1. *Browsing*: The customer is browsing the website without purchasing.
2. *Added to Cart*: The customer has added items to their cart but has not checked out.
3. *Checkout*: The customer is in the checkout process but has not completed the purchase.
4. *Purchased*: The customer has completed a purchase.

We have observed customer behavior and estimated the following transition probabilities for moving from one state to another.

- Customers in the *Browsing* state stay there with 60% probability, move to *Added to Cart* with 30% probability, and go directly to *Checkout* with 10% probability.
- Customers who have added items to their cart have a 20% chance of going back to *Browsing*, a 50% chance of staying in *Added to Cart*, a 20% chance of moving to *Checkout*, and a 10% chance of making a *Purchase*.
- Customers in *Checkout* have a 60% chance of staying in *Checkout*, a 30% chance of making a *Purchase*, and a 10% chance of returning to *Added to Cart*.
- Once in the *Purchased* state, customers stay there permanently (100% probability), as the process ends with a purchase.

The goal is to estimate the probability that a customer will complete a purchase within 6 steps, starting from the *Browsing* state. Write down the transition matrix and use the `MarkovChainGen` function in a Monte Carlo loop to estimate the the purchase probability in 6 steps. Write a Python code and report the result.

Exercise 9.14. You are running a startup company. Let V_t denote the valuation of your company at time t, $t = 0, 1, \ldots$ (say, in months). If V_t, then the company goes bankrupt and stops operating; if $V_t = v_{\max}$, then the company is acquired by a larger company, you receive a payout $v_{\max}$, and the company stops operating. In each time period that the company operates, you invest additional fixed amount c_{invest} in the company, and you also incur an operating cost c_{operate}. If $0 < V_t < v_{\max}$ then

$$V_{t+1} = \begin{cases} V_t + \delta, & \text{with probability } p \\ V_t - \delta, & \text{with probability } 1-p \end{cases}$$

Here $\delta > 0$ is a given parameter. The initial valuation V_0 is an integer multiple of δ, as is $v_{\max}$, so all V_t are also integer multiples of δ. With this model, you will eventually either go bankrupt or be acquired. Consider this model with the following parameter instances:

$$\delta = 2M \text{ sek}, \quad V_0 = 10M \text{ sek}, \quad v_{\max} = 100M \text{ sek},$$
$$c_{\text{operate}} = 10K \text{ sek}, \quad c_{\text{invest}} = 200K \text{ sek}, \quad p = 0.6$$

First, identify the type of random process and explain why. Then design a Monte-Carlo algorithm (write a pseudo-code) to estimate:

a) the probability that the startup goes bankrupt,

b) the expected time until the startup goes bankrupt or is acquired,

c) the expected profit, if the startup is acquired,

d) the expected loss, if the startup goes bankrupt.

Finally, implement your algorithm in a Python environment (with $N = 5000$ Monte Carlo simulations) and report all the above estimations.

Note that, *profit* is the payout, when the company is acquired, minus the initial value of company, minus the total operating cost, minus the total of any investments made. *Loss* is the initial value of company plus the total operating cost plus the total of any investments made, when the company goes bankrupt.

Exercise 9.15. Apply the Gillespie algorithm to solve the predator-prey model

$$R \xrightarrow{\alpha} 2R$$
$$R + F \xrightarrow{\beta} 2F$$
$$F \xrightarrow{\gamma} \emptyset$$

with rates $\alpha = 1$, $\beta = 0.005$, and $\gamma = 0.6$ and initial value $(F_0, R_0) = [50, 100]$ up to final time $t_{final} = 30$. There exists also the deterministic model

$$\frac{dF}{dt} = \beta FR - \gamma F$$
$$\frac{dR}{dt} = \alpha R - \beta FR$$

with $R(0) = R_0$ and $F(0) = F_0$ associated to this simple ecology. Use the ODE solver `solve_ivp` from the `scipy.integrate` library to solve this ODE with the same input data and compare the results of stochastic and deterministic models.

Exercise 9.16. Consider the Michaelis-Menten model

$$S + E \underset{c_2}{\overset{c_1}{\rightleftharpoons}} C \xrightarrow{c_3} P + E$$

which is the standard model for enzyme catalysts. Here, c_1 (forward rate), c_2 (reverse rate), and c_3 (catalytic rate) denote the constant rates of the reactions. Our intention is to solve this model using the Gillespie's algorithm (SSA).

Write down propensity functions and state-change vectors for this model. Assume that for a certain enzyme the reactions rates are $c_1 = 0.002$ mol^{-1}sec^{-1}, $c_2 = 0.1$ sec^{-1} and $c_3 = 0.75$ sec^{-1} where sec stands for the unit of time and mol is the unit for number of proteins. Furthermore, assume that at time $t = 0.1$, sec the number of proteins have been computed as $E(t) = 300$ mol, $S(t) = 200$ mol, $C(t) = 100$ mol, and $P(t) = 50$ mol. The task is to compute the number of proteins in the next time level $t + \tau$. To this aim, we have generated two uniform random numbers $u_1 = 0.64$ and $u_2 = 0.83$ from the $\mathcal{U}(0, 1)$ distribution. The number u_1 must be used to determine the steplength τ, and u_2 to determine the specific reaction that will occur. Given these conditions, proceed to compute the next time level and the number of proteins at this new time. Write down all steps and details of your solution.

Exercise 9.17. Estimate $\mathbb{E}_f[h(X_1, X_2)]$ using the Metropolis-Hastings algorithm where $g(x, y) = xy$ and f is given in (8.5). Use different values $N = 10^3, 10^4, 10^5$ and 10^6.

Exercise 9.18. In thins exercise, we extend the the scenario in Example 8.2 to a two-parameter case where we estimate the recovery rates for two different patient groups, $\theta = (\theta^{(1)}, \theta^{(2)})$. Each parameter, $\theta^{(1)}$ and $\theta^{(2)}$, stands for the recovery rate for a different group of patients and both are assumed to follow exponential distributions with independent Gamma priors. We have two sets of observed recovery times

$$\text{data}_1 = \{5, 8, 12, 7, 9, 10, 3, 6, 8, 11\}$$
$$\text{data}_2 = \{10, 14, 7, 11, 13, 8, 15, 9, 10, 16\}.$$

Each group has 10 observations, and we assume that recovery times for each group are independent and exponentially distributed:

$$p(x|\theta^{(1)}) = \theta^{(1)} e^{-\theta^{(1)} x} \quad \text{and} \quad p(x|\theta^{(2)}) = \theta^{(2)} e^{-\theta^{(2)} x}.$$

Both recovery rates, $\theta^{(1)}$ and $\theta^{(1)}$, are independent and follow Gamma prior

$$\theta^{(i)} \sim \mathcal{G}am(\alpha_i, \beta_i), \quad i = 1, 2.$$

Assume that the prior parameters are $\alpha_1 = 2$, $\beta_1 = 1$, $\alpha_2 = 2$, and $\beta_2 = 1$ which indicates that we expect similar rates for both groups.

For independent recovery times in each group, the likelihood function for each set of obser-

vations is

$$p(\text{data}_i|\theta^{(i)}) = \prod_{j=1}^{10} \theta^{(i)} e^{-\theta^{(i)} x_j}, \quad i = 1, 2,$$

and the joint likelihood of observing all data given $\theta = (\theta^{(1)}, \theta^{(2)})$ is

$$p(\text{data}|\theta^{(1)}, \theta^{(2)}) = p(\text{data}_1|\theta^{(1)}) \cdot p(\text{data}_2|\theta^{(2)}).$$

Start with an initial values $\theta_0 = (\theta_0^{(1)}, \theta_0^{(2)}) = (1, 1)$ and use the random walk sampler with covariance matrix

$$\Sigma = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}$$

at each time step t to generate the new parameter vector θ_{t+1} using the Metropolis-Hastings algorithm. Plot the histogram of generated samples, and compute the mean, variance and the confidence intervals for each parameter.

A Appendix

In this appendix section we summarize some basic definitions and results from probability theory. For more details, see standard textbooks in the subject. As examples I refer you to [**DeGroot-Schervish:2007**] and [**Rubinnstein-Kroese:2017**].

A.1 Random experiments

An experiment whose outcome can not be determined in advance is called a *random experiment*. The *sample space* of the random experiment is the set of all its possible outcomes. We denote the sample space by Ω. For example, assume that a fair coin is flipped three times. If H and T stand for 'heads' and 'tails', the sample space of this experiment is

$$\Omega = \{HHH, HHT, HTH, THH, TTH, THT, HTT, TTT\}$$

which contains eight possible outcomes. Here THT means that the first flip lands tails, the second heads, and the third tails. Subspaces of the sample space are called *events*. For example the event A that the second flip is heads is

$$A = \{HHH, HHT, THH, THT\}.$$

We say that event A *occurs* if the outcome of the experiment is one of the elements of A. Since events are sets, we can apply the usual set operations to them. For example, the event

$$A \cup B$$

is the event that A or B or both occur, and the event

$$A \cap B$$

is the event that A and B both occur. Similar notation holds for unions and intersections of more than two events. For intersection of n events $A_1, A_2, \ldots, A_n$, we usually use the abbrevia-

tion $A_1A_2 \dots A_n = A_1 \cap A_2 \cap \cdots \cap A_n$, for simplicity. The event A^c called the complement of A, is the event that A does not occur. Two events A and B are called *disjoint* if their intersection is empty.

Definition A.1. The probability $\mathbb{P}$ is a rule that assigns a number $\mathbb{P}(A)$ to each event $A \subseteq \Omega$ such that

1. $0 \leqslant \mathbb{P}(A) \leqslant 1$,
2. $\mathbb{P}(\Omega) = 1$,
3. for any sequence $A_1, A_2, \dots$ of disjoint events we have

$$\mathbb{P}\left(\bigcup_k A_k\right) = \sum_k \mathbb{P}(A_k). \tag{A.1}$$

The item 1 states that the probability that the outcome of the experiment lies within A is some number between 0 and 1. The item 2 states that with probability 1 any outcome is a member of the sample space Ω, and the item 3 states that for any set of mutually disjoint events, the probability that at least one of these events occurs is equal to the sum of their respective probabilities.

Since A and A^c are always mutually disjoint, and since $A \cup A^c = \Omega$, we have from items 2 and 3 that

$$1 = \mathbb{P}(\Omega) = \mathbb{P}(A \cup A^c) = \mathbb{P}(A) + \mathbb{P}(A^c)$$

or equivalently

$$\mathbb{P}(A^c) = 1 - \mathbb{P}(A).$$

In other words, the probability that an event does not occur is 1 minus the probability that it does.

In the coin flipping experiment, since the coin is fair, the eight possible outcomes are equally likely to occur and thus has probability 1/8. For example $\mathbb{P}(\{HTH\}) = 1/8$. Since each event A is the union of the events $\{HHH\}, \dots, \{TTT\}$, and this events are disjoint, we have

$$\mathbb{P}(A) = \frac{|A|}{|\Omega|}$$

where $|A|$ denotes the number of outcomes in A. For example, the probability of the event A that the second flip is heads is $\mathbb{P}(A) = 4/8 = 1/2$.

A.2 Conditional probability and independence

Assume that $B \subset \Omega$ is an event. Given that the outcome lies in B, the event A will occur if and only if $A \cap B$ occurs and the relative chance of A occurring is therefore $\mathbb{P}(A \cap B)/\mathbb{P}(B)$. This leads to the definition of the *conditional probability* of A given B:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}. \tag{A.2}$$

For example, if a fair coin is flipped 3 times, and B is the event of total number of heads being 2, then the probability of event A that the second flip is heads given that B occurs is $\frac{2/8}{3/8} = \frac{2}{3}$ because

$$B = \{HHT, HTH, THH\}, \quad \mathbb{P}(B) = \frac{3}{8}$$

and

$$A = \{HHH, HHT, THH, THT\}, \quad A \cap B = \{HHT, THH\}, \quad \mathbb{P}(A \cap B) = \frac{2}{8}.$$

From (A.2), by changing the role of A and B we can write

$$\mathbb{P}(AB) = \mathbb{P}(A)\mathbb{P}(B|A), \tag{A.3}$$

and this formula can be generalized for any sequence of events $A_1, A_2, \ldots, A_n$,

$$\mathbb{P}(A_1 A_2 \cdots A_n) = \mathbb{P}(A_1)\mathbb{P}(A_2|A_1)\mathbb{P}(A_3|A_1A_2)\cdots\mathbb{P}(A_n|A_1\cdots A_{n-1}) \tag{A.4}$$

which is known as the *product rule of probability*.

Assume that $B_1, B_2, \ldots, B_n$ are disjoin events and their union is Ω. Then any event $A \subset \Omega$ can be written as $A = \cup_{k=1}^{n}(A \cap B_k)$. From the third property of probability, i.e. (A.1), we have $\mathbb{P}(A) = \sum_{k=1}^{n} \mathbb{P}(A \cap B_k)$. Then form (A.2) we can write

$$\mathbb{P}(A) = \sum_{k=1}^{n} \mathbb{P}(A|B_k)\mathbb{P}(B_k), \tag{A.5}$$

which is known as the *law of total probability*. Then, from the fact that $\mathbb{P}(A)\mathbb{P}(B_j|A) = \mathbb{P}(AB_j) = \mathbb{P}(A|B_j)\mathbb{P}(B_j)$, we may write

$$\mathbb{P}(B_j|A) = \frac{\mathbb{P}(A|B_j)\mathbb{P}(B_j)}{\sum_{k=1}^{n} \mathbb{P}(A|B_k)\mathbb{P}(B_k)} \tag{A.6}$$

which is known as the *Bayes' rule*.

Two events A and B are called *independent* if $\mathbb{P}(A|B) = \mathbb{P}(A)$ which means that the occurrence of B does not effect on the occurrence of A. An equivalent definition is: A and B are independent if and only if

$$\mathbb{P}(AB) = \mathbb{P}(A)\mathbb{P}(B).$$

Thus definition can be extended to a sequence of events.

Definition A.2. The events $A_1, A_2, \ldots,$ are called independent if for any k and any distinct indexes $i_1, \ldots, i_k$ we have

$$\mathbb{P}(A_{i_1} A_{i_2} \cdots A_{i_k}) = \mathbb{P}(A_{i_1})\mathbb{P}(A_{i_2})\cdots\mathbb{P}(A_{i_k}).$$

A.3 Random variables and distributions

It might not always be feasible or necessary to provide a model for a random experiment through a detailed description of Ω and $\mathbb{P}$. In practice, we are only interested in certain observations in the experiment. These quantities of interest that are determined by the results of the experiment are known as *random variables*, which are usually denoted by capital letters

X, Y or Z with or without subscripts.

Example A.1. Consider an experiment in which a fair coin is tossed n times. In this experiment, the sample space Ω can be regarded as the set of outcomes consisting of the 2^n different sequences of elementary outcomes, for instance $\{\underbrace{HTHHT\cdots T}_{n \text{ times}}\}$. Assume that we are interested only in the number of heads in the observed outcome. Let X be a real-valued function defined on Ω that counts the number of heads in each outcome. For example, for $n = 10$ in the elementary outcome $s = HHTTTHTTTH$, we have $X(s) = 4$. For each possible sequence s, the value $X(s)$ equals the number of heads in the sequence. The possible values for the function X are $\{0, 1, \ldots, 10\}$.

Definition A.3 (Random Variable)**.** Let Ω be the sample space for an experiment. A real-valued function X that is defined on Ω is called a random variable.

The *cumulative distribution function (cdf)*, or more simply the distribution function, F of a random variable X is defined for any real number x by

$$F(x) = \mathbb{P}(X \leqslant x).$$

A random variable that can take either a finite or at most a countable number of possible values is said to be a *discrete random variable*. For a discrete random variable X we define its *probability mass function (pmf)* $f(x)$ by

$$f(x) = \mathbb{P}(X = x).$$

If X is a discrete random variable that takes on one of the possible values $x_1, x_2, \ldots$, then, since X must take one of these values, we have

$$\sum_{k=1}^{\infty} f(x_k) = 1.$$

Example A.2. Assume that a biased coin with p the probability of heads is flipped n times. Suppose that we are interested only in number of heads in this experiment. The number of heads is a random variable, let us denote it by X, and can take any of the values in $\{0, 1, \ldots, n\}$. Each elementary event $\{HTH\cdots T\}$ with exactly k heads and $n-k$ tails has probability $p^k(1-p)^{n-k}$, and there are $\binom{n}{k}$ such events. Thus we have

$$f(k) = \mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \ldots, n.$$

This is the pmf of random variable X. The cdf of X then is

$$F(k) = \sum_{j=0}^{k} \mathbb{P}(X = j) = \sum_{j=0}^{k} \binom{n}{j} p^j (1-p)^{n-j}, \quad k = 0, 1, \ldots, n.$$

A random variable X is said to have a continuous distribution if there exists a positive

function f with total integral 1, such that for all a and b,

$$\mathbb{P}(a \leqslant X \leqslant b) = \int_a^b f(u)\mathrm{d}u.$$

The function f is called the *probability density function (pdf)* of X. Note that in the continuous case the cdf is given by

$$F(x) = \mathbb{P}(X \leqslant x) = \int_{-\infty}^{x} f(u)\mathrm{d}u.$$

Differentiating both sides yields

$$\frac{d}{\mathrm{d}x}F(x) = f(x).$$

Note that in the discrete case we use the term probability mass function (pmf) for f and in the continuous case the term probability density function (pdf). In a more advance probability theory, both pmf and pdf can be viewed as particular instances of a general notion called *probability density*. Therefore, from here on we will call f a pdf in both discrete and continuous cases.

We use the notation $X \sim f$ and $X \sim F$ to denote that X has the pdf f and cdf F, respectively. Sometimes we write f_X to stress that f is the distribution of random variable X. In Tables 1 and 2 the list of some well-known discrete and continuous distributions are given.

Table 1: Some well-known discrete distributions

Name	Notation	$f(x)$	domain of x	Parameters
Bernoulli	$\mathcal{B}er(p)$	$p^x(1-p)^{1-x}$	$\{0,1\}$	$0 \leqslant p \leqslant 1$
Binomial	$\mathcal{B}in(n,p)$	$\binom{n}{x}p^x(1-p)^{n-x}$	$\{0,1,\ldots,n\}$	$0 \leqslant p \leqslant 1$
Discrete uniform	$\mathcal{DU}\{1,\ldots,n\}$	$\frac{1}{n}$	$\{1,\ldots,n\}$	$n \in \{1,\ldots,n\}$
Geometric	$\mathcal{G}eo(p)$	$p(1-p)^{1-x}$	$\{1,2,\ldots\}$	$0 \leqslant p \leqslant 1$
Poisson	$\mathcal{P}oi(\lambda)$	$\mathrm{e}^{-\lambda}\frac{\lambda^x}{x!}$	$\mathbb{N}$	$\lambda > 0$

Table 2: Some well-known continuous distributions

Name	Notation	$f(x)$	domain of x	Parameters
Uniform	$\mathcal{U}(a,b)$	$\frac{1}{b-a}$	$[a,b]$	$a < b$
Normal	$\mathcal{N}(\mu,\sigma^2)$	$\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$\mathbb{R}$	$\sigma > 0$, $\mu \in \mathbb{R}$
Gamma	$\mathcal{G}am(\alpha,\beta)$	$\frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}\mathrm{e}^{-\beta x}$	$\mathbb{R}_+$	$\alpha,\beta > 0$
Exponential	$\mathcal{E}xp(\lambda)$	$\lambda\mathrm{e}^{-\lambda x}$	$\mathbb{R}_+$	$\lambda > 0$
Inv-Gamma	$\mathcal{I}nv\mathcal{G}am(\alpha,\beta)$	$\frac{\beta^\alpha}{\Gamma(\alpha)}(1/x)^{\alpha+1}\mathrm{e}^{-\beta/x}$	$\mathbb{R}_+$	$\alpha,\beta > 0$
Beta	$\mathcal{B}eta(\alpha,\beta)$	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$	$[0,1]$	$\alpha,\beta > 0$
Weibull	$\mathcal{W}eib(\alpha,\lambda)$	$\frac{\alpha}{\lambda}(\frac{x}{\lambda})^{\alpha-1}\mathrm{e}^{-(x/\lambda)^\alpha}$	$\mathbb{R}_+$	$\alpha,\lambda > 0$

A.4 Expectation and variance

The distribution of a random variable X contains all of the probabilistic information about X. Summaries of the distribution, such as expected value and variance, can be useful for giving people some information about X without trying to describe the entire distribution.

The intuitive idea of the *expectation* or *mean* of a random variable is that it is the weighted average of the possible values of the random variable with the weights equal to the probabilities.

Definition A.4. Let X be a random variable with pdf f. The expectation (or expected value or mean) of X, denoted by $\mathbb{E}[X]$ (or sometimes μ), is defined by

$$\mathbb{E}[X] = \begin{cases} \sum_x x f(x), & \text{discrete case} \\ \int_{-\infty}^{\infty} x f(x) \mathrm{d}x, & \text{continuous case,} \end{cases}$$

provided that the sum and integral in the definition are finite.

Example A.3. Let X be a random variable with Bernoulli distribution with parameter p, that is, assume that X takes only the two values 0 and 1 with $\mathbb{P}(X = 1) = p$. Then the mean of X is

$$\mathbb{E}[X] = 0 \times (1 - p) + 1 \times p = p.$$

If X has exponential distribution with parameter λ then

$$\mathbb{E}[X] = \int_0^{\infty} \lambda x e^{-\lambda x} dx = \frac{1}{\lambda}.$$

It is important to note that although $\mathbb{E}[X]$ is called the expectation of X, it depends only on the distribution of X. Every two random variables that have the same distribution will have the same expectation. So, we refer to the expectation of a distribution even if we do not have in mind a random variable with that distribution.

If X is a random variable then a function of X such as X^2 or $\sin(X)$ is another random variable. The expectation of a function of X, say $g(X)$, is simply defined as

$$\mathbb{E}[g(X)] = \mathbb{E}_f[g(X)] = \begin{cases} \sum_x g(x) f(x), & \text{discrete case} \\ \int_{-\infty}^{\infty} g(x) f(x) \mathrm{d}x, & \text{continuous case,} \end{cases} \tag{A.7}$$

provided that the sum and integral are finite. Here f is the pdf of X.

Another useful quantity is the variance which measures the spread or dispersion of the distribution.

Definition A.5. The **variance** of a random variable X is denoted by $\mathrm{Var}(X)$ (or σ^2) and is defined by

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

The square root of the variance is called *standard deviation*.

The variance tells us how much X deviates from its mean value. If X has infinite mean or if the mean of X does not exist, we say that $\mathrm{Var}(X)$ does not exist. In Table 3 the expectations and variances of some well-known distributions are summarized.

Table 3: Expectations and variances of some well-known distributions

Name	Notation	$\mathbb{E}(X)$	$\mathrm{Var}(X)$
Bernoulli	$\mathcal{B}er(p)$	p	$p(1-p)$
Binomial	$\mathcal{B}in(n,p)$	np	$np(1-p)$
Geometric	$\mathcal{G}eo(p)$	$1/p$	$(1-p)/p^2$
Poisson	$\mathcal{P}oi(\lambda)$	λ	λ
Uniform	$\mathcal{U}(a,b)$	$(a+b)/2$	$(b-a)^2/12$
Normal	$\mathcal{N}(\mu,\sigma^2)$	μ	σ^2
Gamma	$\mathcal{G}am(\alpha,\beta)$	α/β	α/β^2
Exponential	$\mathcal{E}xp(\lambda)$	$1/\lambda$	$1/\lambda^2$
Inv-Gamma	$\mathcal{I}nv\mathcal{G}am(\alpha,\beta)$	$\frac{\beta}{\alpha-1}$, $\alpha>1$	$\frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$, $\alpha>2$
Beta	$\mathcal{B}eta(\alpha,\beta)$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(1+\alpha+\beta)}$
Weibull	$\mathcal{W}eib(\alpha,\lambda)$	$\lambda\Gamma(1+1/\alpha)$	$\lambda^2\left[\Gamma(1+2/\alpha)-\Gamma(1+1/\alpha)^2\right]$

A.5 Joint distribution

Let $X_1, X_2, \ldots, X_d$ be random variables describing some random experiments. We can collect them to a random vector $X = (X_1, X_2, \ldots, X_d)$. Then the joint distribution of $X_1, \ldots, X_d$ is specified by the joint cdf

$$F(x_1, \ldots, x_d) = \mathbb{P}(X_1 \leqslant x_1, \ldots, X_d \leqslant x_d)$$

and the joint pdf is given in the discrete case by

$$f(x_1, \ldots, x_d) = \mathbb{P}(X_1 = x_1, \ldots, X_d = x_d),$$

and in the continuous case, f is such that

$$\mathbb{P}(X \in B) = \int_B f(x_1, \ldots, x_d)\mathrm{d}x_1 \ldots \mathrm{d}x_d$$

for any measurable set B in $\mathbb{R}^d$. The marginal pdfs can be recovered from the joint pdf by integration or summation. For example in the case of continuous random vector (X, Y) with joint pdf $f(x, y)$, the pdf f_X of X is obtained as

$$f_X(x) = \int f(x, y)\mathrm{d}y.$$

Suppose that X and Y are both discrete or both continuous, with joint pdf f, and suppose that $f_X(x) > 0$. Then the conditional pdf of Y given $X = x$ is given by

$$f_{Y|X}(y|x) = \frac{f(x, y)}{f_X(x)}, \quad \text{for all } y.$$

The corresponding conditional expectation is (in the continuous case)

$$\mathbb{E}[Y|X = x] = \int y f_{Y|X}(y|x)\mathrm{d}y.$$

Note that $\mathbb{E}[Y|X = x]$ is a function of x, say $h(x)$. The corresponding random variable $h(X)$ is written as $\mathbb{E}[Y|X]$.

When the conditional distribution of Y given X is identical to that of Y, X and Y are said to be independent. More precisely:

Definition A.6 (Independent Random Variables)**.** The random variables $X_1, \ldots, X_d$ are called independent if we have

$$\mathbb{P}(X_1 \in A_1, \ldots, X_d \in A_d) = \mathbb{P}(X_1 \in A_1) \times \cdots \times \mathbb{P}(X_d \in A_d).$$

for all events $\{X_i \in A_i\}$ with $A_i \subset \mathbb{R}$, $i = 1, \ldots, d$.

A direct consequence of the definition above for independence is that the random variables $X_1, \ldots, X_d$ with a joint pdf f (discrete or continuous) are independent if and only if

$$f(x_1, \ldots, x_d) = f_{X_1}(x_1) \cdots f_{X_d}(x_d) \tag{A.8}$$

for all $x_1, \cdots, x_d$, where f_{X_k} are the marginal pdfs.

An infinite sequence $X_1, X_2, \ldots$ of random variables is called independent if for any finite choice of parameters $i_1, i_2, \ldots, i_d$ (none of them the same) the random variables $X_{i_1}, \ldots, X_{i_d}$ are independent. Random variables $X_1, X_2, \ldots$ that are *independent and identically distributed (iid)* are frequently appeared in probabilistic models.

The expectation of any real-valued function g of variables $X_1, \ldots, X_d$ is defined as

$$\mathbb{E}[g(X_1, \ldots, X_d)] = \int \cdots \int g(x_1, \ldots, x_d) f(x_1, \ldots, x_d) \mathrm{d}x_1 \cdots \mathrm{d}x_d.$$

A direct consequent of the definitions of expectation and independence (equation (A.8)) is

$$\mathbb{E}[a + b_1 X_1 + b_2 X_2 + \cdots + b_d X_d] = a + b_1 \mathbb{E}[X_1] + b_2 \mathbb{E}[X_2] + \cdots + b_d \mathbb{E}[X_d]$$

for any sequence of independent random variables $X_1, \ldots, X_d$ and constant a, b_1, ..., b_d. For independent random variables we also have (prove!)

$$\mathbb{E}[X_1 X_2 \cdots X_d] = \mathbb{E}[X_1]\mathbb{E}[X_2] \cdots \mathbb{E}[X_d].$$

Sometimes it is useful to have a summary of how much the two random variables depend on each other. The *covariance* and *correlation* are additional notions to measure that dependence, but they only capture a linear dependence.

Definition A.7. The covariance of two random variables X and Y with expectations $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$ is defined as

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)].$$

A scaled version of the covariance is given by correlation coefficient

$$\rho(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

where $\sigma_X^2 = \mathrm{Var}(X)$ and $\sigma_Y^2 = \mathrm{Var}(Y)$.

It can be shown that the correlation coefficient is always lies between -1 and 1. Values close to 0 show a more dependency while values close to 1 and -1 stand for a linear independency between two variables. Some important properties of the variance and covariance are listed bellow.

1. $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$,

2. $\mathrm{Var}(aX+b) = a^2\mathrm{Var}(X)$,
3. $\mathrm{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}(X)\mathbb{E}(Y)$,
4. $\mathrm{Cov}(X,Y) = \mathrm{Cov}(Y,X)$,
5. $\mathrm{Cov}(aX+bY,Z) = a\mathrm{Cov}(X,Z) + b\mathrm{Cov}(Y,Z)$,
6. $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X,Y)$,
7. X and Y independent $\Rightarrow \mathrm{Cov}(X,Y) = 0$.

For random vectors, such as $X = (X_1, \ldots, X_d)^T$, it is convenient to write the expectations and covariances in vector notation as

$$\mathbb{E}(X) = (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_d])^T = (\mu_1, \ldots, \mu_d)^T = \mu$$
$$\Sigma = \mathbb{E}[(X-\mu)(X-\mu)^T] = [\mathrm{Cov}(X_i, X_j)]_{i,j=1,\ldots,d}.$$

Note that any covariance matrix Σ is a symmetric positive semidefinite matrix.

A.6 Functions of random variables

Suppose that X is a random variable and $Z = g(X)$ for some monotonically increasing function g. To find the pdf of Z from that of X we first write

$$F_Z(z) = \mathbb{P}(Z \leqslant z) = \mathbb{P}(X \leqslant g^{-1}(z)) = F_X(g^{-1}(z)).$$

Differentiation with respect to z then gives

$$f_Z(z) = f_X(g^{-1}(z))\frac{d}{\mathrm{d}z}g^{-1}(z) = \frac{f_X(g^{-1}(z))}{g'(g^{-1}(z))} = \frac{f_X(x)}{g'(x)}.$$

For monotonically decreasing functions, $\frac{d}{\mathrm{d}z}g^{-1}(z)$ in the first equation need to be replaced with its negative value. For example let $Z = aX + b$ where $a \neq 0$ and suppose that $a > 0$. Since $g^{-1}(z) = \frac{1}{a}(z-b)$ and $g'(z) = a$, we have $f_Z(z) = \frac{1}{a}f_X(\frac{1}{a}(z-b))$. Similarly, for $a < 0$ we have $f_Z(z) = \frac{1}{-a}f_X(\frac{1}{a}(z-b))$. Thus in general

$$f_Z(z) = \frac{1}{|a|}f_X\left(\frac{z-b}{a}\right).$$

Now consider a random vector $X = (X_1, \ldots, X_d)^T \in \mathbb{R}^d$, and let $Z = AX$ where A is an $m \times d$ matrix. Then Z is a random vector in $\mathbb{R}^m$. If we know the joint distribution of X, then we can derive the joint distribution of Z. First, we can show that

$$\mu_Z = A\mu_X, \quad \Sigma_Z = A\Sigma_X A^T$$

because $\mu_Z = \mathbb{E}[Z] = \mathbb{E}[AX] = A\mathbb{E}[X] = A\mu_X$ and $\Sigma_Z = \mathbb{E}[(Z-\mu_Z)(Z-\mu_Z)^T] = \mathbb{E}[A(X-\mu_X)(A(X-\mu_X))^T] = A\mathbb{E}[(X-\mu_X)(X-\mu_X)^T]A^T = A\Sigma_X A^T$. Then by using some tools from calculus, we can prove that

$$f_Z(z) = \frac{1}{|\det(A)|}f_X(x). \tag{A.9}$$

In a more general case for a fixed $x \in \mathbb{R}^m$, let $z = g(x)$ where $g : \mathbb{R}^d \to \mathbb{R}^m$ is invertible. Then

$$f_Z(z) = \frac{1}{|\det J_g(g(x))|}f_X(x),$$

where J_g is the Jacobian matrix of g.

A.7 Joint normal random variables

Assume that X has standard normal distribution, i.e., $X \sim \mathcal{N}(0,1)$. Then X has density f_X given by

$$f_X(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \quad x \in \mathbb{R}.$$

If $Z = \mu + \sigma X$ then $\mathbb{E}[Z] = \mu$ and $\mathrm{Var}(Z) = \sigma^2$, i.e., $Z \sim \mathcal{N}(\mu, \sigma^2)$, thus Z has density function

$$f_Z(z) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2\right), \quad z \in \mathbb{R}.$$

We can also state this as follows: if $Z \sim \mathcal{N}(\mu, \sigma^2)$ then

$$\frac{Z-\mu}{\sigma} \sim \mathcal{N}(0,1),$$

which is called *standardization*. This can be generalized to d dimensions. Let $X_1, \ldots, X_d$ be independent and standard normal random variables. The joint pdf of X is given by

$$f_X(x) = (2\pi)^{-n/2} \exp\left(-\frac{1}{2}x^T x\right), \quad x \in \mathbb{R}^d.$$

If $Z = \mu + BX$ for some $m \times d$ matrix B then Z has expectation vector μ and covariance matrix $\Sigma = BB^T$. The random vector Z is said to have a *jointly normal* or *multivariate normal* distribution. We write

$$Z \sim \mathcal{N}(\mu, \Sigma).$$

In a special case when B is $d \times d$ and invertible, the pdf of Z can be derived as follows. If $Y = Z - \mu$ then from (A.9) the pdf of Y is given by

$$f_Y(y) = \frac{(2\pi)^{-n/2}}{|\mathrm{det(B)}|} \exp\left(-\frac{1}{2}(B^{-1}y)^T(B^{-1}y)\right).$$

Using the facts that $|\mathrm{det(B)}| = \sqrt{|\mathrm{det}(\Sigma)|}$ and $(BB^T)^{-1} = \Sigma^{-1}$, we have

$$f_Y(y) = \frac{1}{\sqrt{(2\pi)^d |\mathrm{det}(\Sigma)|}} \exp\left(-\frac{1}{2}y^T\Sigma^{-1}y\right).$$

Since $Z = Y + \mu$ for a constant vector μ, we have $f_Z(z) = f_Y(z-\mu)$, and therefore

$$f_Z(z) = \frac{1}{\sqrt{(2\pi)^d |\mathrm{det}(\Sigma)|}} \exp\left(-\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right).$$

Conversely, given a covariance matrix $\Sigma = (\sigma_{ij})$, there exists a unique lower triangular matrix

$$B = \begin{bmatrix} b_{11} & 0 & \cdots & 0 \\ b_{21} & b_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

such that $\Sigma = BB^T$. Recall that every covariance matrix is positive semi-definite. The lower triangular matrix B can be obtained efficiently via the *Cholesky factorization*.

A.8 Generating normal random variables

In this section we use the *inverse transform method* to generate normal random variables. Before start reading this part, read Section 3 if you are not familiar with the inverse transform method. The problem is that the inverse of the normal cdf is not available explicitly, and it is numerical expensive to compute. A suitable transformation helps to make things simpler. Assume that $X \sim \mathcal{N}(\mu, \sigma^2)$. The pdf of X is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right), \quad x \in \mathbb{R}. \tag{A.10}$$

It is enough to generate from the standard normal distribution $\mathcal{N}(0,1)$ because any random variable $Z \sim \mathcal{N}(\mu, \sigma^2)$ can be written as $Z = \sigma X + \mu$ where $X \sim \mathcal{N}(0,1)$. The cdf of the standard normal distribution then is

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}u^2\right) \mathrm{d}u = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

where erf is the error function. The error function should be computed via its series representation or numerical integration. Computing an accurate inverse for F is numerically expensive thus the inverse transform method is not practically applicable.

One of the earliest method for generating from the standard normal distribution was developed by Box and Muller as follows: Assume that $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$ are two independent random variables. The variable (X, Y) has joint distribution

$$f_{X,Y}(x,y) = \frac{1}{2\pi} \exp\left(-\frac{1}{2}(x^2+y^2)\right), \quad (x,y) \in \mathbb{R}^2.$$

By transferring the variables to polar coordinates (r, θ) with change of variables

$$x = r\cos\theta, \quad y = r\sin\theta, \quad r \geqslant 0,\ \theta \in [0, 2\pi), \tag{A.11}$$

the joint distribution of the transferred variables (R, Θ) becomes

$$f_{R,\Theta}(r,\theta) = \frac{r}{2\pi} \exp\left(-\frac{r^2}{2}\right), \quad r \geqslant 0,\ \theta \in [0, 2\pi),$$

where the factor r behind the exponential term comes from the determinant of the Jacobian matrix of the transformation. Now, we have

$$F_{R,\Theta}(r,\theta) = \int_0^{\theta}\int_0^{r} \frac{r'}{2\pi} \exp\left(-\frac{r'^2}{2}\right) dr' d\theta' = -\frac{\theta}{2\pi} \exp\left(-\frac{r^2}{2}\right),$$

which shows that R and Θ are independent with cdf's

$$F_\Theta(\theta) = -\frac{\theta}{2\pi},\ \theta \in [0, 2\pi), \quad F_R(r) = \exp\left(-\frac{r^2}{2}\right),\ r \geqslant 0.$$

Consequently, $\Theta \sim \mathcal{U}(0, 2\pi)$ and R has the same distribution as $\sqrt{Q}$ with $Q \sim \mathcal{E}xp(1/2)$ because

$$F_{\sqrt{Q}}(r) = \mathbb{P}(\sqrt{Q} \leqslant r) = \mathbb{P}(Q \leqslant r^2) = \exp\big(-r^2/2\big).$$

Both R and Θ are easy to generate. First we generate $U_1 \sim \mathcal{U}(0,1)$ and $U_2 \sim \mathcal{U}(0,1)$. Then we set $\Theta = 2\pi U_1$ and $R = \sqrt{-2\ln U_2}$ from (3.2). Finally, we transfer R and Θ back to X and Y via transformation (A.11):

$$X = R\cos(\Theta) = \sqrt{-2\ln U_2}\cos(2\pi U_1),$$
$$Y = R\sin(\Theta) = \sqrt{-2\ln U_2}\sin(2\pi U_1).$$

Note that, using this approach we generate two standard random variables X and Y from two uniform variables U_1 and U_2. A Python code is given here.

```
def RandNormal(mu, sigma2, N):
    U1 =  np.random.rand(N)
    U2 =  np.random.rand(N)
    X = np.sqrt(-2*np.log(U1))*np.cos(2*np.pi*U2)
    X = mu + np.sqrt(sigma2)*X
    return X
```

The code snippet below generates N random normal points and plots the corresponding histograms. Outputs are shown in Figure 30 for $N = 500$ and 5000.

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize = (5,3))
N = 500
X = RandNormal(0, 1, N)
plt.hist(X, bins = 30, histtype = 'bar', color = 'red', density = 'true')
x = np.linspace(-4,4,200)
f = 1/(np.sqrt(2*np.pi))*np.exp(-x**2/2)
plt.plot(x,f,linestyle = '-', color = 'blue')
plt.title('Histogram of $X$ and the pdf $f(x)$')
plt.xlabel('$X$'); plt.ylabel('Frequency %')
```

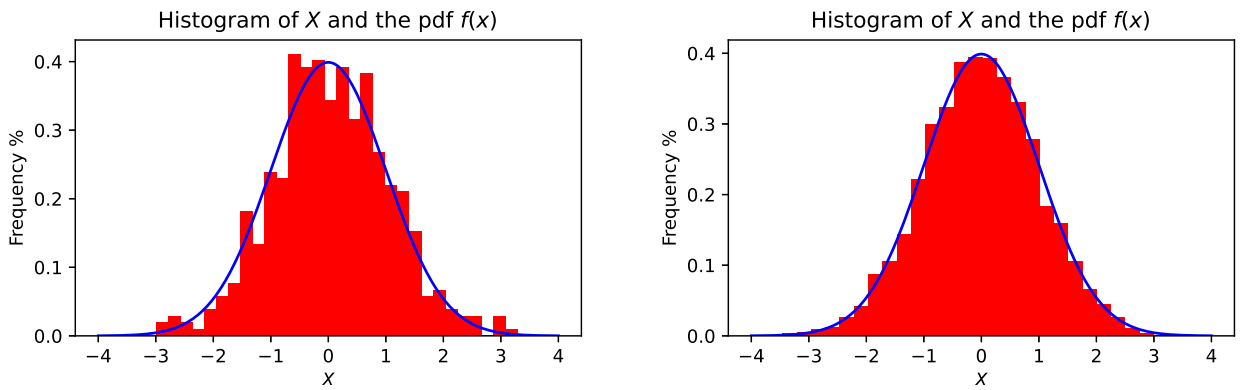


Figure 30: Histograms of generated random points from the standard normal distribution with $N = 500$ (left), $N = 5000$ (right)

A.9 Generating from multivariate distributions

Before start reading this section, you may need to take a look at Section 3 for random point generation algorithms for univariate distributions.

To generate a random vector $X = (X_1, \ldots, X_d)$ for a given d dimensional distribution with pdf $f(x) = f(x_1, \ldots, x_d)$, we can use the product rule (A.4)

$$f(x_1, \ldots, x_d) = f_1(x_1) f_2(x_2|x_1) \cdots f_d(x_d|x_1, \cdots, x_{d-1})$$

where $f_1(x_1)$ is the marginal pdf of X_1, $f_k(x_k|x_1, \ldots, x_{k-1})$ is the conditional pdf of X_k given $X_1 = x_1, \ldots, X_{k-1} = x_{k-1}$. If $X_1, X_2, \ldots, X_d$ are independent then $f_k(x_k|x_1, \ldots, x_{k-1}) = f_k(x_k)$, and the techniques of univariate distributions can be simply applied to each component individually. To generate X in the case that variables are dependent, one can first generate X_1 from pdf $f_1(x_1)$, then given $X_1 = x_1$ generate X_2 from $f_2(x_2|x_1)$, and so on. To run this approach, a knowledge on conditional distributions is required. Markov models provide a feasible and simple way to obtain such a knowledge and to generate from a general joint distribution. See section 6.

Drawing from the multi-normal distribution

Assume that $X \sim \mathcal{N}(\mu, \Sigma)$, where $\mu = (\mu_1, \ldots, \mu_d)^T$ is the mean vector and $\Sigma \in \mathbb{R}^{d \times d}$ is the covariance matrix of X. Then we have $X = \mu + BZ$ where Z is a vector of iid random variables with distributions $\mathcal{N}(0, 1)$, and B is the Cholesky factorization of the covariance matrix Σ. The following Python code can be used to generate `N` multi-normal random variables with expectation vector `mu` and covariance matrix `Sigma`.

```
def RandMultiNormal(mu, Sigma, N):
    dim = np.size(mu)
    Z = X = np.zeros([dim,N])
    if dim > 1:
       B = np.linalg.cholesky(Sigma)
    else:
       B = [np.sqrt(Sigma)]
    for d in range(dim):
        Z[d,:] = np.random.normal(0, 1, N)
    X = np.matlib.repmat(mu, N, 1).T + np.matmul(B,Z)
    return X
```

A.10 Limit theorems

In this part we review the *law of large numbers* and the *central limit theorem*. Both are associated with sums of independent random variables.

Let $X_1, X_2, \ldots$ be iid random variables with expectation μ and variance σ^2. For each n, let

$$S_n = X_1 + X_2 + \cdots + X_n.$$

Since $X_1, X_2, \ldots$ are iid, we have $\mathbb{E}[S_n] = n\mathbb{E}[X_1] = n\mu$ and $\mathrm{Var}(S_n) = n\mathrm{Var}(X_1) = n\sigma^2$. The law of large numbers states that S_n/n is close to μ for large n. Here is the more precise statement.

Theorem A.8 (Strong Law of Large Numbers)**.** If $X_1, \ldots, X_n$ are iid with expectation μ, then

$$\mathbb{P}\left(\lim_{n\to\infty} \frac{S_n}{n} = \mu\right) = 1.$$

The central limit theorem describes the limiting distribution of S_n (or S_n/n), and it applies to both continuous and discrete random variables. Loosely, it states that the random sum S_n has a distribution that is approximately normal, when n is large. The more precise statement is given next.

Theorem A.9 (Central Limit Theorem)**.** If $X_1, \ldots, X_d$ are iid with expectation μ and variance $\sigma^2 < \infty$, then for all $x \in \mathbb{R}$,

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \leqslant x\right) = \Phi(x)$$

where Φ is the cdf of the standard normal distribution.

In other words, S_n has a distribution that is approximately normal, with expectation $n\mu$ and variance $n\sigma^2$. These theorem are valid independent of the type of distribution of $X_1, X_2, \ldots$.

There is also a central limit theorem for random vectors. The multidimensional version is as follows: Let $X_1, \ldots, X_n$ be iid random vectors with expectation vector μ and covariance matrix Σ. Then for large values of n the random vector $X_1 + \cdots + X_n$ has approximately a multivariate normal distribution with expectation vector $n\mu$ and covariance matrix $n\Sigma$.

References

[1] M. H. DeGroot, M. J. Schervish, *Probability and Statistics*, 4th Edition, Pearson Education, Inc., 2012.

[2] S. M. Ross, *Simulation*, Academic Press, 3rd Edition, 2002.

[3] R. Y. Rubinnstein, D. P. Kroese, *Simulation and the Monte Carlo Method*, 3rd Edition, Wiley, 2017.

[4] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods,* Springer-Verlag, New York, 2nd edition, 2004.