Overview and Motivation
Introduction to Monte Carlo Methods

The term *Monte Carlo* is typically associated with the process of modeling and simulating a system affected by randomness: Several random scenarios are generated, and relevant statistics are gathered in order to assess, e.g., the performance of a decision policy or the value of an asset. Stated as such, it sounds like a fairly easy task from a conceptual point of view, even though some programming craft might be needed. Although it is certainly true that Monte Carlo methods are extremely flexible and valuable tools, quite often the last resort approach for overly complicated problems impervious to a more mathematically elegant treatment, it is also true that running a bad Monte Carlo simulation is very easy as well. There are several reasons why this may happen:

- We are using a wrong model of uncertainty:
  - Because we are using an unrealistic probability distribution
  - Or because we are missing some link among the underlying risk factors
  - Or because some unknown parameters have been poorly estimated
  - Or because the very nature of uncertainty in our problem does not lend itself to a stochastic representation

- The output estimates are not reliable enough, i.e., the estimator variance is so large that a much larger sample size is required.

- There is a systematic error in the estimates, which could be low or high biased.

- The way we generate sample paths, possibly discretizing a continuous-time model, induces a non-negligible error.

- We are using poor random variate generators.

- There is some possibly subtle bug in the computer program implementing the method.

Some of these issues are technical and can be addressed by the techniques that we will explore in this book, but others are more conceptual in nature and point out a few intrinsic and inescapable limitations of Monte Carlo methods; it is wise not to forget them, while exploring the richness and power of the approach.
The best countermeasure, in order to avoid the aforementioned pitfalls, is to build reasonably strong theoretical foundations and to gain a deep understanding of the Monte Carlo approach and its variants. To that end, a good first step is to frame Monte Carlo methods as a numerical integration tool. Indeed, while the term simulation sounds more exciting, the term Monte Carlo sampling is often used. The latter is more appropriate when we deal with Monte Carlo sampling as a tool for numerical integration or statistical computing. Granted, the idea of simulating financial markets is somewhat more appealing than the idea of computing a multidimensional integral. However, a more conceptual framework helps in understanding powerful methods for reducing, or avoiding altogether, the difficulties related to the variance of random estimators. Some of these methods, such as low-discrepancy sequences and Gaussian quadrature, are actually deterministic in nature, but their understanding needs a view integrating numerical integration and statistical sampling.

In this introductory chapter, we consider first the historical roots of Monte Carlo; we will see in Section 1.1 that some early Monte Carlo methods were actually aimed at solving deterministic problems. Then, in Section 1.2 we compare Monte Carlo sampling and Monte Carlo simulation, showing their deep relationship. Typical simulations deal with dynamic systems evolving in time, and there are three essential kinds of dynamic models:

- Continuous-time models
- Discrete-time models
- Discrete-event models

These model classes are introduced in Section 1.3, where we also illustrate how their nature affects the mechanics of Monte Carlo simulation. In this book, a rather relevant role is played by applications involving optimization. This may sound odd to readers who associate simulation with performance evaluation; on the contrary, there is a multiway interaction between optimization and Monte Carlo methods, which is outlined in Section 1.4.

In this book we illustrate a rather wide range of applications, which may suggest the idea that Monte Carlo methods are almost a panacea. Unfortunately, this power may hide many pitfalls and dangers. In Section 1.5 we aim at making the reader aware of some of these traps. Finally, in Section 1.6 we list a few software tools that are commonly used to implement Monte Carlo simulations, justifying the choice of R as the language of this book, and in Section 1.7 we list prerequisites and references for readers who may need a refresher on some background material.

1.1 Historical origin of Monte Carlo simulation

Monte Carlo methods involve random sampling, but the actual aim is to estimate a deterministic quantity. Indeed, a well-known and early use of Monte Carlo-
like methods is Buffon’s needle approach to estimate $\pi$.\footnote{The estimation of $\pi$ by Monte Carlo methods will also be used as an illustration of variance reduction strategies in Chapter 8.} The idea, illustrated in Fig. 1.1, is to randomly throw $n$ times a needle of length $l$ on a floor consisting of wood strips of width $t > l$, and to observe the number of times $h$ that the needle crosses the border between two strips. Let $X$ be the distance from the center of the needle to the closest border; $X$ is a uniform random variable on the range $[0, t/2]$, and its probability density function is $2/t$ on that interval and 0 outside.\footnote{In Chapter 3 we offer a refresher on common probability distributions, among other things.} By a similar token, let $\theta$ be the acute angle between the needle and that border; this is another uniformly distributed variable, on the range $[0, \pi/2]$, with density $2/\pi$ on that support. Using elementary trigonometry, it is easy to see that the needle will cross a border when

$$X \leq \frac{l}{2} \sin \theta.$$  

The probability of this event is found by integrating the joint density of the random variables $X$ and $\theta$ over the relevant domain. Assuming independence, the joint density is just the product of the two marginals. Therefore, we have to compute a double integral as follows:

$$\int_0^{\pi/2} \int_0^{l/2} \frac{4}{t\pi} \sin \theta \, dx \, d\theta = \frac{2l}{t\pi} \int_0^{\pi/2} \sin \theta \, d\theta = \frac{2l}{t\pi}.$$  

If we want to estimate $\pi$, we can use a sample estimate of this probability, i.e., the ratio of $h$ over $n$:

$$\frac{h}{n} \approx \frac{2l}{t\pi} \quad \Rightarrow \quad \hat{\pi} = \frac{2l}{th}.$$
\texttt{estPi <- function(N) \{}
\texttt{x <- 2*runif(N)-1}
\texttt{y <- 2*runif(N)-1}
\texttt{h <- sum(x^2+y^2<=1)}
\texttt{return(4*h/N)}
\}

\textbf{R programming notes:}
\begin{enumerate}
\item A function is created by the \texttt{function} keyword, followed by the input arguments, and assigned to a variable.
\item \texttt{runif} is a typical R function, featuring the \texttt{x} prefix, for generating a random sample of given size from a selected distribution. In the following, we will meet similar functions, like \texttt{exp} and \texttt{norm}, to sample from an exponential and a normal distribution, respectively.
\item The expression \texttt{sum(x^2+y^2<=1)} illustrates an interesting feature of R to get rid of for loops: It returns a vector of elements with values \texttt{FALSE} or \texttt{TRUE}, which are interpreted in the sum as 0/1, respectively.
\end{enumerate}

\section*{FIGURE 1.2 Using R to estimate \(\pi\).}

We see that the original problem is purely deterministic, but we use Monte Carlo sampling as an estimation tool.

A slightly more practical approach, taking advantage of R, is illustrated in the code of Fig. 1.2. The idea is shooting \(n\) random bullets on the square \([-1,1] \times [-1,1]\) and evaluating the fraction \(h/n\) of points falling inside the unit circle \(x^2+y^2=1\). Since the ratio between the area of the circle and the area of the square is \(\pi/4\), we have

\[\hat{\pi} = \frac{4h}{n}.\]

\textbf{Running the code,}\footnote{Later, we will understand in more detail what is accomplished by the command \texttt{set.seed(55555)}. For now, it is enough to say that we always use this function to set the state of pseudorandom number generators to a given value, so that the reader can replicate the experiments and obtain the same results. There is nothing special in the argument 55555, apart from the fact that 5 is the author's favorite number.} we obtain

\begin{verbatim}
> set.seed(55555)
> estPi(1000)
[1] 3.132
> estPi(1000)
[1] 3.14
> estPi(1000)
[1] 3.084
> estPi(1000000)
[1] 3.141388
\end{verbatim}
> \[ \pi \]

[1] 3.141593

Clearly, a huge sample size is needed to obtain a fairly accurate estimate, and this is certainly not the smartest way to estimate \( \pi \).

In more recent times, in the period between 1930 and 1950, Monte Carlo methods were used by the likes of Enrico Fermi, Stanislaw Ulam, and John von Neumann to solve problems related to physics. Indeed, such methods were used in the 1950s when the hydrogen bomb was developed at the Los Alamos laboratories. It was then that the term *Monte Carlo* was coined after the well-known gambling house. Then, luckily enough, the idea moved to other domains, including operations research and economics. At present, Monte Carlo methods are extremely widespread in many domains, including risk management and financial engineering.

## 1.2 Monte Carlo simulation vs. Monte Carlo sampling

The terms *simulation* and *sampling* are both used with reference to Monte Carlo methods. Indeed, there is no 100% sharp line separating the two concepts, and a couple of examples are the best way to get the point.

### Example 1.1 Shortfall probability in wealth management

Consider an investor who allocates her wealth \( W_0 \) to \( n \) risky financial assets, such as stock, bonds, derivatives, and whatnot. At time \( t = 0 \), the asset price of asset \( i, i = 1, \ldots, n \), is \( P_i^0 \), and the (integer) number of shares of assets \( i \) in the portfolio is \( h_i \). Therefore, the initial wealth is

\[
W_0 = \sum_{i=1}^{n} h_i P_i^0.
\]

The portfolio is held up to time \( t = T \), and the price of asset \( i \) at the end of this investment horizon depends on a set of underlying risk factors, such as the term structure of interest rates, spreads due to credit risk, inflation, oil price, the overall state of the economy, etc. Let \( X \) be a vector random variable, with joint density \( f_X(x) \), representing these factors. If we assume that the price of each asset at time \( T \) is given by a function of the underlying factors,

\[
P_i^T(X),
\]

then the terminal wealth is a function of \( X \),

\[
W_T = \sum_{i=1}^{n} h_i P_i^T(X).
\]
which is itself a random variable. On the one hand, we are interested in the expected value of \( W_T \). However, since risk is a quite relevant issue, we might also be interested in the shortfall probability with respect to a target wealth \( H \):

\[
P\{W_T < H\}.
\]

Whatever the case, we are led to the expected value of some function \( g(X) \):

\[
\int g(x)f_X(x)\,dx. \tag{1.1}
\]

If we choose \( g(X) = W_T \) we obtain the expected value of future wealth; if we choose the indicator function of the event \( \{W_T < H\} \), i.e.,

\[
g(X) = 1_{\{W_T < H\}} = \begin{cases} 1, & \text{if } W_T < H, \\ 0, & \text{otherwise}, \end{cases}
\]

then we obtain the shortfall probability. Other risk measures can be selected and will be discussed in Chapter 13. Evaluating the multidimensional integral in Eq. (1.1) is quite difficult, even when we know the analytical form of the involved functions. As we shall see, by using Monte Carlo we may draw a sample of \( m \) observations \( X_{k}, k = 1, \ldots, m \), and estimate the quantities of interest. We are using random sampling as a tool for numerical integration.

In the example above, no simulation is involved at all, assuming that we are able to sample directly the random variable \( X \) given its density. In other cases, we may have to cope with a more complicated dynamic model that describes the dynamics of the underlying risk factors over the time interval \([0, T]\). In such a case, we may not be able to sample risk factors directly, and we have to generate a set of sample paths. Dynamic models may be represented by continuous-time stochastic differential equations\(^4\) or by relatively simple discrete-time processes as in the following example.

\section{Example 1.2 An autoregressive process}

Time series models are quite often used in financial econometrics to describe the evolution of a quantity of interest over time. A simple

\[^{4}\text{Stochastic differential equations are introduced in Section 3.7, and approaches for sample path generation are described in Chapter 6.}\]
example of this class of models is an autoregressive (AR) process of order 1 like

\[ X_t = X_{t-1} + \epsilon_t, \quad (1.2) \]

where \( X_0 \) is given and \( \epsilon_t \) is the driving stochastic process, i.e., a sequence of random variables \( \epsilon_t, \ t = 1, 2, 3, \ldots \). In the simplest models, this noise term consists of a sequence of mutually independent and identically distributed normal random variables. We use the notation \( X \sim N(\mu, \sigma^2) \) to state that \( X \) is a normal random variable with expected value \( \mu \) and variance \( \sigma^2 \). Unfolding the recursion, we immediately see that

\[ X_t = X_0 + \sum_{\tau=1}^{t} \epsilon_{\tau}, \]

which implies

\[ X_t \sim N(X_0, t \sigma^2). \]

The process described by Eq. (1.2) is very easy to analyze, but it is not the most general AR process of order 1. A more general example is

\[ X_t = \mu + \alpha X_{t-1} + \epsilon_t, \]

whose properties critically depend on the coefficient \( \alpha \) (see Example 1.4). In other cases, we may have a system of such equations, with mutually correlated random driving factors and a more complicated structure, preventing us from obtaining the probability distribution of the variable of interest in an explicit form. In such a case, we may have to simulate the dynamical system.

On the basis of the above examples, we may argue that the term sampling could be reserved to those cases in which there is no dynamics to be simulated over time, whereas simulation entails the generation of sample paths. We may need to generate sample paths because the dynamical model is complicated and prevents an analytical solution, or because it is important to check the underlying variables over time. For instance, when pricing a European-style option,\(^5\) we just need to sample the underlying variable at the option maturity. In the easy case of geometric Brownian motion (GBM for short) this boils down to sampling a lognormal random variable, but we have to resort to sample path generation when we step outside that safe and comforting domain, and we enter the realm populated by stochastic volatilities and price jumps. On the contrary, when we are pricing an American-style option, featuring early exercise opportunities, we need a whole sample path, whatever price model we adopt.

\(^5\)See Section 3.9 and Chapter 11.
As is usually the case, also the above distinction is not so clear-cut. In both settings we are actually evaluating a possibly high-dimensional integral of a function, in order to estimate a probability or an expectation. The function may be given in analytical form, or it may be defined by a possibly complicated black box, requiring several lines of code to be evaluated. Conceptually, there is no difference. Indeed, numerical integration plays such a pivotal role for a full appreciation of Monte Carlo methods that we will devote the whole of Chapter 2 to it.

1.3 System dynamics and the mechanics of Monte Carlo simulation

When we have to describe the dynamic evolution of a system over time, we typically resort to one of the following model classes:

- Discrete-time models
- Continuous-time models
- Discrete-event models

From a technical point of view, as usual, the distinction is not sharp. For instance, even if we choose a continuous-time model, we have to discretize it in some way for the sake of computational feasibility. This implies that we actually simulate a discrete-time approximation, but given the issues involved in sample path generation, which involves nontrivial concepts related to the numerical solution of stochastic differential equations, it is better to stick to the classification above. It is also possible to create somewhat hybrid models, such as stochastic processes including both a diffusion component, which is continuous, and jumps, which are related to discrete-event dynamics.

1.3.1 DISCRETE-TIME MODELS

In discrete-time models we assume that the time horizon $[0, T]$ is discretized in time intervals (time buckets) of width $\delta t$. In principle, nothing forbids nonuniform time steps, which may actually be required when dealing with certain financial derivatives; nevertheless, we usually stick to the uniform case for the sake of simplicity. The system is actually observed only at time instants of the form

$$k \delta t, \quad k = 0, 1, \ldots, M,$$

where $T = M \delta t$. What happens between two discrete-time instants is not relevant. Typically, we forget about the time bucket size $\delta t$, which could be a day, a week, or a month, and we use discrete-time subscripts, like $i = 0, 1, 2, \ldots$, directly.
### Example 1.3 Cash on hand

Consider the cash management problem for a retail bank. During each day $t$, we observe cash inflows $w_t^+$ and cash outflows $w_t^-$, corresponding to cash deposits and withdrawals by customers, respectively. Hence, if we denote by $H_t$ the cash holding at the end of day $t$, we have the dynamic equation

$$H_t = H_{t-1} + w_t^+ - w_t^-.$$

Such a model makes sense if we are only interested in the balance at the end of each day.

The AR model of Example 1.2 is a very basic example of time series models. From a technical point of view, simulating a time series model is usually no big deal.

### Example 1.4 Simulating a simple AR process

Let us consider the following autoregressive process:

$$X_t = \mu + \alpha X_{t-1} + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$. The R function in Fig. 1.3 can be used for its simulation. In the specific case

$$X_t = 8 + 0.8X_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0, 1), \quad (1.3)$$

we run the function as follows:

```r
> set.seed(55555)
> AR(40, 0.8, 8, 1, 50)
```

obtaining the plot in Fig. 1.4.

In the above examples we had the purely random evolution of a system. In a more general setting, we may have the possibility of influencing, i.e., of partially controlling its dynamics. For instance, in discrete-time stochastic optimization models, we deal with a system modeled by the state transition equation

$$s_{t+1} = g_t(s_t, x_t, \epsilon_{t+1}), \quad t = 0, 1, 2, \ldots, \quad (1.4)$$

where

- $s_t$ is a vector of state variables, observed at the end of time period $t$. 

AR <- function(X0,a,mu,sigma,T){
    X <- rep(0,T+1)
    X[1] <- X0
    for (t in 1:T){
        X[t+1] <- mu+a*X[t]+rnorm(1,0,sigma)
    }
    plot(0:T,X,pch=15)
    lines(0:T,X)
}

R programming notes:

1. In order to allocate memory to an array, we use `rep(0,T+1)` here; as an alternative, we may also use `numeric(T+1)`.

2. The syntax of for loops is self-explanatory. Braces are used to enclose multiple statements; in this case, since there is only one statement inside the loop, they could be omitted.

3. The function `plot` is used here to lay down dots, with a character specified by the optional `pch` parameter; the function `lines` connects the dots with lines, without opening another window.

FIGURE 1.3 R code to simulate a simple autoregressive process.

FIGURE 1.4 Sample path of the AR process of Eq. (1.3).
• $x_t$ is a vector of control variables, i.e., decisions made after observing the state $s_t$.

• $\epsilon_{t+1}$ is a vector of disturbances, realized after we apply the control variable $x_t$.

If the sequence of disturbances $\epsilon_t$ is intertemporally independent, we obtain a Markov process. Markov processes are relatively easy to represent, simulate, and control. We should also observe that the state need not be a continuous variable. We may have a system with discrete states, resulting in a discrete-time Markov chain. As an example, we may consider a Markov chain modeling the migration in the credit rating of a bond issue. Sometimes, we use a discrete-state Markov chain to approximate a continuous-state system by aggregation: We partition the state space with a mutually exclusive and collectively exhaustive collection of subsets and identify each one with a single representative value.

1.3.2 CONTINUOUS-TIME MODELS

The typical representation of a continuous-time model relies on differential equations. We are all familiar with the differential equation $F = ma$ from elementary physics, where $F$ is force, $m$ is mass, and $a$ is acceleration, i.e., the second-order derivative of position with respect to time. As a more financially motivated example, imagine that you deposit an amount of cash $B(0) = B_0$, at time $t = 0$, in a safe bank account. If $r$ is the continuously compounded interest rate on your deposit, the evolution in time of wealth over time can be represented as

$$\frac{dB(t)}{dt} = rB(t),$$  \hspace{1cm} (1.5)

whose solution, given the initial condition $B(0) = B_0$, is

$$B(t) = B_0 e^{rt}.$$  \hspace{1cm} (1.6)

If the interest rate is not constant, and it is given by $r(t)$, we obtain

$$B(t) = B(0) \exp \left\{ \int_0^t r(\tau) \, d\tau \right\}.$$  \hspace{1cm} (1.6)

This solution is easily obtained by rewriting Eq. (1.5) in the differential form

$$\frac{dB}{B} = r(t) \, dt,$$

which, by recalling the derivative of the natural logarithm, can be transformed to

$$d \log B = r(t) \, dt.$$  

\hspace{1cm} (1.6)

\hspace{1cm} (1.6)

$^6$The Markov property is essential in dynamic programming, as we shall see in Chapter 10; we will cover some material on Markov chains in Chapter 14.

$^7$See Section 3.1.3 for further details and a motivation of such an interest rate.
Integrating on the interval \([0, \delta t]\) and taking the exponential to get rid of the logarithm immediately yields Eq. (1.6). In less lucky cases, we have to resort to numerical methods, which typically yield a discrete-time approximation of the continuous-time system.

Monte Carlo methods come into play when randomness is introduced into the differential equation, yielding a stochastic differential equation. A well-known model in this vein is the GBM

\[
\text{d}S(t) = \mu S(t) \text{d}t + \sigma S(t) \text{d}W(t),
\]

which is arguably the most elementary model used to represent the random evolution of stock prices. With respect to the previous case, we have an additional term \(W(t)\), which is a Wiener process. We will deal later with the underpinnings of this kind of stochastic process,\(^8\) but for now suffice to say that the increment of the Wiener process over a finite time interval \([t, t + \delta t]\) is a normal random variable:

\[
\delta W(t) = W(t + \delta t) - W(t) \sim \mathcal{N}(0, \delta t).
\]

Loosely speaking, the differential \(\text{d}W(t)\) is the limit of this increment for \(\delta t \to 0\). Thus, we might consider simulating random sample paths of GBM by a straightforward discretization scheme:

\[
\delta S(t) = S(t + \delta t) - S(t) = \mu S(t) \delta t + \sigma S(t) [W(t + \delta t) - W(t)].
\]

This is called the Euler discretization scheme, and it immediately leads to the discrete-time process

\[
S(t + \delta t) = S(t)(1 + \mu \delta t + \sigma S(t)\sqrt{\delta t} \varepsilon(t + \delta t)),
\]

where \(\varepsilon(t + \delta t) \sim \mathcal{N}(0, 1)\). This kind of process, in terms of simulation, is not unlike an AR process and is easy to deal with. Unfortunately, this rather naive discretization suffers from a significant inconvenience: We obtain the wrong probability distribution. Indeed, it is easy to see that we generate normally distributed prices; therefore, at least in principle, we may generate a negative price, which does not make sense for limited liability assets like stock shares.

Using tools from stochastic calculus, we will see that we may solve the stochastic differential equation exactly and find an exact discretization:

\[
S(t + \delta t) = S(t) \exp \left[ \left( \mu - \frac{\sigma^2}{2} \right) \delta t + \sigma \sqrt{\delta t} \varepsilon(t + \delta t) \right].
\]

This results in a lognormal distribution, ruling out negative prices. We also notice that \(\mu\) is related to the drift of the process, i.e., its average rate of change over time, whereas \(\sigma\) plays the role of a volatility.

\(^8\)The Wiener process is introduced in Chapter 3, and its sample path generation is discussed in Chapter 6.
1.3 System dynamics and the mechanics of Monte Carlo simulation

\[ \text{simGBM} <- \text{function} \ (S0, \mu, \sigma, T, \text{numSteps}, \text{numRepl}) \{ \]
\[ \quad \text{dt} <- \frac{T}{\text{numSteps}} \]
\[ \quad \mu_T <- (\mu - \sigma^2/2)\times \text{dt} \]
\[ \quad \sigma_T <- \sqrt{\text{dt}} \times \sigma \]
\[ \quad \text{pathMatrix} <- \text{matrix}(\text{ncol}=\text{numRepl}, \text{ncol}=\text{numSteps}+1) \]
\[ \quad \text{pathMatrix}[1,1] <- S0 \]
\[ \quad \text{for} \ (i \ \text{in} \ 1: \text{numRepl}) \{ \]
\[ \quad \quad \text{for} \ (j \ \text{in} \ 2: (\text{numSteps}+1)) \{ \]
\[ \quad \quad \quad \text{pathMatrix}[i,j] <- \]
\[ \quad \quad \quad \text{pathMatrix}[i,j-1] \times \exp(\text{rnorm}(1, \mu_T, \sigma_T)) \]
\[ \quad \quad \} \]
\[ \quad \}\]
\[ \text{return}(\text{pathMatrix}) \]

R programming notes:

1. In the first few lines we precompute invariant quantities; it is always a good rule to avoid useless recalculations in for loops.
2. Note how the function matrix is used to preallocate a matrix with a given number of rows and columns; in this book we always store sample paths in rows, rather than columns, and include the initial value in the first column.
3. The rnorm function is used to generate a sample from a normal distribution with given parameters.
4. This function is not optimized in the sense that it is not vectorized for efficiency. We will see an alternative implementation in Chapter 6.

---

**FIGURE 1.5** R code to generate sample paths of geometric Brownian motion.

### Example 1.5 Simulating geometric Brownian motion

The R code in Fig. 1.5 can be used to generate sample paths of GBM. The function returns a matrix whose rows correspond to numRepl sample paths starting from initial state S0, over numSteps time steps from 0 to T, for a GBM with parameters \( \mu \) and \( \sigma \). The Rsnapshot of Fig. 1.6 yields the plots in Fig. 1.7. We immediately appreciate the role of the volatility \( \sigma \).

This solution is exact in the sense that there is no discretization error in the sample path generation; we are left with sampling errors that are typical of Monte Carlo methods. In other cases, things are not that easy: Even though, in the end, continuous-time models may boil down to discrete-time models, they do have some peculiarities, which may be hard to deal with.
set.seed(55555)
S0 <- 40
mu <- 0.25
sigma <- 0.1
sigma2 <- 0.4
T <- 1
numSteps <- 50
numRepl <- 10
Paths1 <- simGBM(S0, mu, sigma, T, numSteps, numRepl)
Paths2 <- simGBM(S0, mu, sigma2, T, numSteps, numRepl)
maxVal <- max(Paths1, Paths2)
minVal <- min(Paths1, Paths2)
labx <- bquote(italic(t)) # how to generate plot labels
laby <- bquote(italic(S(t))) # with italic fonts
par(mfrow = c(1,2))
plot(Paths1[,1], type="l", ylab=laby, xlab=labx,
     main="Vol. 10%", ylim=c(minVal, maxVal))
for (k in 2:10) lines(Paths1[,k])
plot(Paths2[,1], type="l", ylab=laby, xlab=labx,
     main="Vol. 40%", ylim=c(minVal, maxVal))
for (k in 2:10) lines(Paths2[,k])
par(mfrow = c(1,1))

R programming notes:

1. Note the use of variables minVal and maxVal to set an appropriate vertical range for the plot, which is then controlled by the ylim parameter of plot. You may also use the range function to gather the smallest and largest element in an array.

2. The function plot opens a new window, whereas lines draws a line on the same window.

3. In order to control the layout of several plots on the same window, you may use par(mfrow = c(1,2)); it is advisable to reset the layout to the default value after plotting.

FIGURE 1.6 R script to plot GBM sample paths for different volatilities.

1.3.3 DISCRETE-EVENT MODELS

The GBM sample paths depicted in Fig. 1.7 look noisy enough, but continuous. Indeed, it can be shown that they are continuous, in a stochastic sense, but nondifferentiable everywhere. The sample paths of a discrete-event dynamic system are quite different in nature. It is important to avoid a possible confusion between discrete event and discrete time. Discrete-event systems are continuous-time models and, unlike discrete-time models, there is no constant time step \( \delta t \). Like discrete-time systems, the state changes only at some time
FIGURE 1.7 Sample paths of geometric Brownian motion for different levels of volatility.

instants, corresponding to the occurrence of some event, but the time between events is in general a continuous random variable.

Perhaps, the simplest discrete-event model, to be further discussed in Section 3.2.2, is the Poisson process. This is an instance of a counting process: The value \( N(t) \) counts the number of events that occurred during the time interval \([0, t]\), where the times \( X_k \) elapsing between events \( k-1 \) and \( k, k = 1, 2, 3, \ldots \), are a sequence of independent and identically distributed exponential variables.

By convention, \( X_1 \) is the time at which the first event occurs after the start time \( t = 0 \). A sample path of a Poisson process is illustrated in Fig. 1.8; we see that the process "jumps" whenever an event occurs, so that sample paths are piecewise constant. The underlying exponential distribution is characterized by a single parameter \( \lambda \) that can be interpreted as a "rate," i.e., the average number of events occurring per unit time.\(^9\) This basic Poisson process can be generalized by allowing for a nonconstant rate \( \lambda(t) \), which yields an inhomogeneous Poisson process. We may also allow for arbitrary jump sizes. If we associate a continuous probability distribution with the jump size, we obtain a continuous-state system, whereas unit jumps yield a discrete state space corresponding to integer numbers. This second generalization is called compound Poisson process, which is a discrete-event, continuous-state model.

Financially motivated examples of such models are:

\(^9\) In Section 3.2.2 we shall see that \( 1/\lambda \) is the corresponding expected value of the exponential random variable.
- Models of stock prices allowing for jumps. Pure-jump models have been proposed, but we may also integrate diffusion processes, like GBM, with jumps. Stochastic processes in this more general form are known as Lévy processes.

- Models of credit rating migration, where the rating of a bond issue changes across a finite set of states at random times. This is a discrete-event, discrete-state model. Continuous-time Markov chains are an example in this family.

Discrete-event models are the rule in other application domains, like the simulation of manufacturing systems and logistic networks. These kinds of simulation problems are, in a sense, dual to the simulation problems faced in economics and finance: The mathematics involved is generally trivial, but there is an incredible complexity in data bookkeeping, resulting in huge computer programs, unless one resorts to a special-purpose simulation tool. On the contrary, when dealing with a stochastic differential equation, the program often consists of just a few lines of code, possibly hiding a quite sophisticated mathematical complexity. In the next section we give a simple example of a discrete-event system, so that the reader can appreciate the mechanics involved.

1.3.3.1 Simulation of discrete-event systems: a single queue

Let us consider a very simple system consisting of a single server (e.g., a bank teller), which must satisfy a set of incoming requests for service (e.g., customers entering a bank). The time between arrivals of consecutive customers is ran-

\footnote{The curious reader might refer, e.g., to [4].}
dom; we might assume that it is a plain Poisson process or some time-varying process. The time needed to serve each request is random as well. The system dynamics is not difficult to describe:

- If the arriving customer finds the server idle, it starts her service immediately and will free the resource after service completion; the server state changes from idle to busy.
- If the server is idle, the new customer joins a queue of requests waiting for service. The simplest queuing discipline is first-in, first-out (FIFO), i.e., requests are handled according to the order of their arrivals.
- When the server terminates a service, it starts servicing the first customer in the queue, if any; otherwise, its state changes from busy to idle.

In order to describe the state of this system, we would certainly use \( L(t) = 0, 1, 2, \ldots \), the number of customers in the system, which includes the one being served, if any, and the state of the server, \( S(t) \in \{0, 1\} \), where we assume that 1 corresponds to busy and 0 to idle. We also need some information about the future departure of the currently served customer, if any. A sample path of this single-server queuing system is depicted in Fig. 1.9. Statistics that we might be interested to collect are the average queue length or the average server utilization, which are essentially the integrals of the sample paths in Fig. 1.9 divided by the simulation time horizon. We immediately see that some bookkeeping is required, as these are not the familiar sample means, but rather integrals over time. A performance measure that can be evaluated as a sample mean is the average customer waiting time.

In some specific cases, there is no need to resort to simulation, as the system can be analyzed mathematically. For instance, if the service and the interarrival times are both exponentially distributed, the system is essentially a continuous-time Markov chain, thanks to the memoryless property of the exponential dis-
Such a queue is denoted by $M/M/1$, where the $M$ refers to the memoryless nature of the two stochastic processes involved, customer arrivals and customer service, and the $1$ refers to a system consisting of a single server. If we denote the arrival and service rates by $\lambda$ and $\mu$, it can be shown that

$$W = \frac{\rho}{1 - \rho}, \tag{1.9}$$

where $\rho = \lambda/\mu$ is the utilization rate; clearly, stability requires $\mu > \lambda$, i.e., the service rate, i.e., the average number of requests served per unit time, must be larger than the arrival rate. In more complicated cases, typically involving network of servers, or more difficult distributions, or different queuing disciplines, one may be forced to resort to simulation.

Let us denote by $A_j$, $S_j$, and $W_j$ the arrival time, service time, and waiting time, respectively, for customer $j$. It is possible to find a recursive equation for the waiting time of customer $j$. Customer $j$ has to wait if it arrives before the completion time of customer $j - 1$. Denoting the latter quantity by $C_{j-1}$, we have

$$W_j = \max\{0, C_{j-1} - A_j\}. \tag{1.10}$$

However, the completion time of customer $j - 1$ is

$$C_{j-1} = A_{j-1} + W_{j-1} + S_{j-1}.$$ 

Plugging this into Eq. (1.10), we find

$$W_j = \max\{0, A_{j-1} + W_{j-1} + S_{j-1} - A_j\}
= \max\{0, W_{j-1} + S_{j-1} - I_j\}, \tag{1.11}$$

where $I_j = A_j - A_{j-1}$ is the interarrival time between customers $(j - 1)$ and $j$. We note that, in a more general setting possibly involving priorities, balkings, or multiple servers, complicated data bookkeeping would be needed to track customer waiting times, requiring suitable data structures and a corresponding programming effort. In this lucky case, it is easy to implement the recursion in R, as shown in Fig. 1.10. In Fig. 1.11 we also show an R script running the simulator in order to check the results against the exact formula of Eq. (1.9). With $\rho = 0.90$, which is a fairly high value for the utilization rate, we observe a lot of variability around the correct value:

```
> rho/num/(1-rho)
[1] 9.090909
> MM1_Queue(lambda, mu, 100000)
[1] 9.026053
> MM1_Queue(lambda, mu, 100000)
[1] 10.34844
> MM1_Queue(lambda, mu, 100000)
[1] 8.143615
```

\(^{11}\) See Section 3.2.2.1.
1.3 System dynamics and the mechanics of Monte Carlo simulation

```r
MML_Queue <- function(lambda=1, mu=1.1, howmany=10000)
{
  W <- numeric(howmany)
  for (j in 1:howmany)
    { intTime <- rexp(1, rate=lambda) servTime <- rexp(1, rate=mu) W[j] <- max(0, W[j-1] + servTime - intTime) }
  return(mean(W))
}
```

R programming notes:
1. As usual, the array W is preallocated, using numeric in this case.
2. Note how R offers the possibility of setting default values for input arguments. Hence, the function could be called using just `MML_Queue()`: if you want to assign a different value to the second parameter, you may circumvent parameter positions by naming them when calling the function, as in `MML_Queue(mu=1.2)`.
3. The `rexp` function is used to sample from an exponential distribution. Note that the input argument to this function is the average rate at which events occur, rather than the expected time elapsing between consecutive events.

**FIGURE 1.10** Implementation of Lindley's recursion for a $M/M/1$ queue.

```r
set.seed(55555)  # ensure repeatability
lambda = 1; mu = 1.1
rho = lambda/mu  # utilization is 90.90%
rho/mu/(1-rho)   # exact value
MML_Queue(lambda, mu, 100000)
MML_Queue(lambda, mu, 100000)
MML_Queue(lambda, mu, 100000)
# change parameters
lambda = 1
mu = 2
rho = lambda/mu  # utilization is 50%
rho/mu/(1-rho)   # exact value
MML_Queue(lambda, mu, 100000)
MML_Queue(lambda, mu, 100000)
MML_Queue(lambda, mu, 100000)
```

**FIGURE 1.11** Script to check the function `MML_Queue()`.
For a low utilization rate, such as $\rho = 50\%$, we get more reliable results:

> rho/mu/(1-rho)
> MM1_Queue(lambda, mu, 100000)
> MM1_Queue(lambda, mu, 100000)
> MM1_Queue(lambda, mu, 100000)

We need to quantify the reliability of the above estimates, possibly by confidence intervals, as we show in Chapter 7 on simulation output analysis.

1.4 Simulation and optimization

There is a considerable interplay between Monte Carlo methods and optimization problems. A generic finite-dimensional optimization problem can be stated as

$$\min_{x \in S} f(x).$$

where $x \in \mathbb{R}^n$ is an $n$-dimensional vector collecting the decision variables, $S \subseteq \mathbb{R}^n$ is the feasible set, and $f$ is the objective function. The feasible set $S$ is usually represented by a collection of equalities and inequalities. In recent years, there has been an astonishing progress in the speed and reliability of commercial software tools to solve wide classes of optimization problems. As a general rule that we shall illustrate in more detail in Chapter 10, convex optimization problems, whereby the function $f$ and the set $S$ are both convex, can be efficiently solved. In particular, large-scale linear programming problems can be solved in a matter of seconds or minutes. Other families of convex problems, such as second-order cone programming models, can be solved rather efficiently by relatively recent interior-point methods.

The joint progress in algorithmic sophistication and hardware speed makes optimization modeling a more and more relevant tool for many practical problems, including those motivated by economical or financial applications. Nevertheless, there are classes of problems that still are a very hard nut to crack. They include the following:

1. Nonconvex problems, i.e., either problems with a nonconvex objective function featuring many local optima, or problems with a nonconvex feasible set, like integer programming and combinatorial optimization problems

2. Stochastic optimization problems involving uncertainty in the problem data
3. Dynamic decision problems affected by uncertainty, which preclude the
determination of an optimal static solution, but on the contrary require a
dynamic strategy that adapts decisions to the unfolding of uncertainty.

We will see much more on this kind of models, and the related solution algo-
rithms, in Chapter 10. Here we just want to point out that Monte Carlo methods
play a key role in tackling these problems, which we also illustrate in the next
subsections.

1.4.1 NONCONVEX OPTIMIZATION

In a minimization problem with no constraints on the decision variables, if the
objective function is convex, then a locally optimal solution is also a global one.
This property considerably helps in devising solution algorithms. Equivalently,
maximizing a concave function is relatively easy. This is good news for an
economist wishing to solve a decision problem featuring a concave utility func-
tion or a financial engineer minimizing a convex risk measure. However, there
are cases in which such a nice condition is not met, and the need for global op-
timization methods arises. The following example illustrates a typical function
featuring a large number of local optima.

Example 1.6 The Rastrigin function

The Rastrigin function is a sort of benchmark to test global optimization
algorithms, as well as to illustrate how bad an objective function
can be. In the general \( n \)-dimensional case, the Rastrigin function is
defined as

\[
R_n(x) = A n + \sum_{i=1}^{n} \left[ x_i^2 - A \cos(2 \pi x_i) \right],
\]

where \( A \) is some constant. In a bidimensional case, if we choose
\( A = 10 \), we may boil down the function to

\[
R_2(x) = 20 + x^2 - 10 \cos 2 \pi x + y^2 - 10 \cos 2 \pi y,
\]

depending on the two variables \( x \) and \( y \). A surface plot of the Rastrin-
gin function is displayed in Fig. 1.12, where its nonconvexity can be
appreciated. Another view is offered by the contour plot of Fig. 1.13.
These two plots have been obtained by the R script in Fig. 1.14.

The Rastrigin function does illustrate why nonconvexities make an optimizer's
life difficult, but it definitely looks artificial. To see how nonconvexities may
arise in a practical context, let us consider model calibration in asset pricing.
Suppose that we have a model for the price of \( n \) assets indexed by \( k, k = \]
FIGURE 1.12 Surface plot of Rastrigin function.

FIGURE 1.13 Contour plot of Rastrigin function.
rastrigin <- function(x, y) 20 + x^2 - 10 * cos(2*pi*x) + y^2 - 10 * cos(2*pi*y)
x <- seq(-2, 2, length=100)
y <- x
z <- outer(x, y, rastrigin)
persp(x, y, z, theta=30, phi=35, expand=0.6, 
col='lightgreen', shade=0.1, ltheta=120, 
ticktype='detailed')
contour(x, y, z)

R programming notes:

1. In this function we illustrate one possible use of seq to partition an interval using a given number of points.
2. The functions persp and contour can be used to draw surface and contour plots, respectively. Their output can be customized with plenty of optional parameters.
3. To create those plots, we need to combine the coordinates of the two input vectors x and y in any possible way, and then apply the function rastrigin on them to create a matrix. This is accomplished by the function outer; try outer(1:4, 1:4) to figure out how it works.

FIGURE 1.14 Script to produce the plots of the Rastrigin function.

1, ..., n. The pricing function

\[ \hat{P}_k = F_k(\beta_1, \beta_2, \ldots, \beta_m) \]

predicts the price \( \hat{P}_k \) depending on some specific characteristics of the asset \( k \) itself, and a set of general parameters \( \beta_j, j = 1, \ldots, m \), common to several assets, where \( m << n \). Model calibration requires finding a set of parameters that fit as much as possible a set of observed prices \( P^o_k \). After model calibration, we can move on and price other assets in an internally consistent way. To be concrete, an option pricing model is calibrated on the basis of quoted prices of exchange-traded derivatives, and it can then be used to price an over-the-counter derivative for which a market quote is not available. The model can be used, e.g.,

- By the client of an investment bank to check if the price proposed by the bank for a specifically tailored contract makes sense
- By an arbitrageur to spot price inconsistencies paving the way for arbitrage opportunities
- By a risk manager to predict the impact of changes in the underlying factors on the value of a portfolio

A standard way to tackle the problem is by minimizing the average squared distance between the observed price \( P^o_k \) and the theoretical price \( \hat{P}_k \) predicted
by the model, which leads to the nonlinear least-squares problem:

$$\min \sum_{k=1}^{n} \left[ F_k^o - F_k(\beta_1, \beta_2, \ldots, \beta_m) \right]^2$$ (1.13)

s.t. $\beta \in S,$ (1.14)

where $\beta$ is a vector collecting the parameters and $S$ is a feasible set representing restrictions on parameters, such as non-negativity. While ordinary least-squares problems are trivial to solve, in this case the pricing functions may be nonlinear; hence, there is no guarantee in general that the resulting problem is convex, and a local solver may well get trapped in a locally optimal solution yielding a set of unreasonable parameters. A similar consideration applies in parameter estimation, when a complicated likelihood function is maximized.\textsuperscript{12}

Several global optimization algorithms have been proposed, and some of them do guarantee the global optimality of the reported solution. Unfortunately, these algorithms are typically restricted to a specific class of functions, and the optimality guarantee may require demanding computations. As an alternative, if we are not able to exploit specific structural properties of the problem, and a near-optimal solution is enough, possibly the optimal one but with no proof of its status, we may resort to general-purpose stochastic search algorithms. Among the proposed classes of methods, we mention:

- Pure random sampling of points in the feasible domain
- Using a local optimizer along with random multistart strategies
- Using strategies to escape from local optima, like simulated annealing
- Using strategies based on the evolution of a population of solutions, like genetic algorithms and particle swarm optimization

These methods may also be applied when the complementary kind of nonconvexity arises, i.e., we have to deal with a nonconvex set of feasible solutions, as is the case with combinatorial optimization problems, where we have to explore a finite, but huge discrete set of solutions. The above methods come in a possibly confusing assortment of variants, also including hybrid strategies. The effectiveness of each approach may actually depend on some problem features, most notably the presence or absence of difficult constraints. However, all of them rely on some random exploration behavior, based on Monte Carlo sampling.

### 1.4.2 STOCHASTIC OPTIMIZATION

Most financial problems involve some form of uncertainty. Hence, we may have to tackle an optimization problem where some data defining the objective function or the feasible set are uncertain. We face the difficult task of choosing an

\textsuperscript{12} See Section 4.2.3.
optimal solution \( x^* \) in \( S \) before knowing the value of a vector random variable \( \xi \) taking values on \( \Xi \). If so, two questions arise:

- Is \( x^* \) robust with respect to optimality? In other words, what can we say about the quality of the solution across possible realizations of \( \xi \)?
- Is \( x^* \) robust with respect to feasibility? This is an even thornier issue, as the solution we choose might be infeasible for some particularly unlucky scenarios. For instance, we may have solvency issues for a pension fund.

For now, let us focus only on the first issue and consider a problem of the form

\[
\min_{x \in S} F(x) \equiv \mathbb{E}_{\xi} [f(x, \xi)]. \tag{1.15}
\]

The objective function \( f(x, \xi) \) depends on both decisions \( x \) and random factors \( \xi \), but by taking an expectation with respect to \( \xi \) we obtain a deterministic function \( F(x) \). Assuming that the components of \( \xi \) are jointly continuous and distributed according to a density function \( g_\xi(y) \), with support \( \Xi \), the objective function can be written as

\[
F(x) = \int_{y \in \Xi} f(x, y) g_\xi(y) \, dy.
\]

As we point out in Chapter 2, such a multidimensional integral can be extremely difficult to evaluate for a given \( x \), let alone to optimize in general, even when we are able to prove that \( F(x) \) enjoys such nice properties as continuity, differentiability, or even convexity.

A possible approach is to boil down the integral to a sum, by sampling \( K \) scenarios characterized by realizations \( \xi^k \) and probabilities \( \pi^k, k = 1, \ldots, K \). Then, we solve

\[
\min_{x \in S} \sum_{k=1}^{K} \pi^k f(x, \xi^k), \tag{1.16}
\]

which is a more familiar optimization problem. We note that Monte Carlo sampling may play a key role in optimization as a scenario generation tool. However, crude sampling is typically not a viable solution and the following issues must be tackled:

- How can we generate a limited number of scenarios, in order to capture uncertainty without having to solve a huge and intractable problem?
- How can we prove convergence, i.e., that when the sample size goes to infinity, the solution of problem (1.16) converges to the true optimal solution of problem (1.15)?
- The previous question is actually theoretical in nature and definitely beyond the scope of this book. A more practical, but obviously related question is: How can we assess the quality of the solution for a finite sample size?
How can we assess the robustness of the solution with respect to sampling variability and, possibly, with respect to higher level uncertainty on the parameters defining the uncertainty model itself?

Some of these hard questions will be addressed in this book, when we deal with variance reduction strategies, numerical integration by Gaussian quadrature, and deterministic sampling based on low-discrepancy sequences. If all this does not sound challenging enough, recall that we may also have feasibility issues. What can we do if a solution proves to be infeasible after we observe the true value of the uncertain parameters? More generally, what if the optimization problem is dynamic and we have to make a sequence of decisions while we gather information step by step? One approach to cope with these multistage stochastic problems relies on multistage stochastic programming with recourse, discussed in Chapter 10; sometimes, they can also be tackled by stochastic dynamic programming.

1.4.3 STOCHASTIC DYNAMIC PROGRAMMING

In the previous section we have considered how uncertainty issues may affect an optimization problem, but we disregarded a fundamental point: the dynamics of uncertainty over time. In real life, when making decisions under uncertainty, we do start with a plan, but then we adjust our decisions whenever new pieces of information are progressively discovered. Rather than coming up with a plan to be rigidly followed over time, we pursue a dynamic strategy in which decisions are adapted on the basis of contingencies. This kind of consideration suggests the opportunity of formulating multistage, dynamic stochastic optimization models, whereby we have to find a sequence of decisions \( x_t, t = 0, 1, \ldots, T \), which are not deterministic but depend on the realized future states. As we shall see, there are different approaches to tackle these quite challenging problems, depending on their nature and on our purpose.

- We may be interested in the first decision \( x_0 \), because this is the decision that we have to implement here and now. The next decisions enter into the optimization model in order to avoid an overly myopic behavior, but when time advances, we will solve the model again according to a rolling horizon strategy. Note that this requires the solution of a sequence of multistage models. This approach is common in management science and in some engineering problems, and it is exemplified by multistage stochastic programming models with recourse.

- In other cases, we want to find a policy in feedback form, i.e., a rule mapping the current state \( s_t \), which evolves according to the state transition equations (1.4), to an optimal decision. The mapping \( x_t^* = F_t(s_t) \) can be found by following another approach, based on stochastic dynamic programming.
To see an example in which the second kind of approach may be preferable, consider pricing American-style options.\textsuperscript{13} In that case, we are interested in the expected value of the objective function, resulting from the application of an optimal exercise policy in the future. This requires a strategy, possibly a suboptimal one, to come up with lower bounds on the option value, to be complemented by upper bounds.

Another common kind of problem whereby a dynamic programming approach may be preferable can be found in economics. To illustrate the idea, let us consider a stylized consumption-saving problem.\textsuperscript{14} A decision maker must decide, at each discrete-time instant, the fraction of current wealth that he consumes immediately, deriving a corresponding utility, and the fraction of current wealth that is saved and invested. The return on the investment can be random and depending on the asset allocation decisions. Even if we assume that return is deterministic, we should consider uncertainty in labor income, which determines the level of wealth available for consumption/saving. A possible model is the following:

\[
\begin{align*}
\max \quad & E_t \left[ \sum_{\tau=t}^{T} -\beta^{T-\tau} U(C_\tau) \right] \\
\text{s.t.} \quad & S_\tau = X_\tau - C_\tau, \\
& X_\tau = R_\tau S_{\tau-1} + Y_\tau, \\
& Y_\tau = P_\tau \epsilon_\tau,
\end{align*}
\]

where

- $E_t[\cdot]$ denotes a conditional expectation given the current state at time $t$.
- $U(\cdot)$ is a utility function related to individual preferences.
- $\beta \in (0, 1)$ is a subjective time discount factor.
- $X_\tau$ is the cash on hand (wealth) at time $\tau$, before making the consumption/saving decision.
- $C_\tau$ is the amount of wealth that is consumed.
- $S_\tau$ is the amount of wealth that is saved.
- $R_\tau$ is the return from period $\tau - 1$ to $\tau$; note that this return is usually random and that $R_\tau$ applies to $S_{\tau-1}$, not $S_\tau$.
- $P_\tau$ is the nominal labor income, which may vary over time according to a deterministic or stochastic trajectory.
- $Y_\tau$ is the noncapital income, i.e., the income from labor that depends on the nominal labor income and multiplicative shocks $\epsilon_\tau$.

\textsuperscript{13}See Example 1.8. We discuss how to price American-style options with Monte Carlo methods in Chapter 11.

\textsuperscript{14}For an extensive discussion of such models see, e.g., [5].
As we shall see in Chapter 10, we may solve such a problem by dynamic programming, which is based on a recursive functional equation for the value function

\[ V_t(X_t) = \max_{C_t \geq 0} \left\{ U(C_t) + \beta E_t [V_{t+1}(X_{t+1})] \right\}, \tag{1.17} \]

where \( V_t(X_t) \) is the expected utility if we start the decision process at time \( t \), from a state with current wealth \( X_t \). In the above model, wealth is the only state variable and randomness is due to random financial return and shocks on nominal income. However, if we model income in a more sophisticated way, including permanent and transitory shocks, we may need additional state variables. From the above equation we see that the problem is decomposed stage by stage, and if we are able to find the sequence of value functions, we have just to solve a static optimization problem to find the optimal decision at each step, given the current state. The value function is the tool to trade off immediate utility against future benefits.

Also note that we make our decision after observing the actually realized state, which does not depend only on our decisions, since it is also affected by external random factors. The policy is implicitly defined by the set of value functions \( V_t(\cdot) \) for each time instant. This is definitely handy if we have to carry out an extensive set of simulations in order to investigate the impact of different economic assumptions on the decision maker's behavior, and it is an advantage of dynamic programming over stochastic programming models with recourse. Nevertheless, dynamic programming has some definite limitations on its own, as it relies on a decomposition that is valid only if some Markovian properties hold for the system states, and it is plagued by the aptly called "curse of dimensionality." This refers to the possibly high dimensionality of the state space. If we have only one state variable, it is easy to approximate value functions numerically on a grid, but if there are many state variables, doing so is practically impossible. Actually, other curses of dimensionality may refer to the need to discretize the expectation in Eq. (1.17) and the computational complexity of the optimization required at each step. Nevertheless, there are approximate dynamic programming approaches that do not guarantee optimality, but typically provide us with very good decisions. The exact approach should be chosen case by case, but what they have in common is the role of Monte Carlo methods:

- To generate scenarios
- To learn an optimal policy
- To evaluate the performance of a policy by simulation

### 1.5 Pitfalls in Monte Carlo simulation

Monte Carlo methods are extremely flexible and powerful tools; furthermore, they are conceptually very simple, at least in their naive and crude form. All of these nice features are counterbalanced by a significant shortcoming: They
can be very inefficient, when each single simulation run is computationally expensive and/or a large sample size is needed to obtain reliable estimates. These computational difficulties are technical in nature and can be (at least partially) overcome by clever strategies. However, there are deeper issues that should not be disregarded and are sometimes related to the very meaning of uncertainty.

### 1.5.1 TECHNICAL ISSUES

Each Monte Carlo replication yields an observation of a random variable, say \( X_k \), for \( k = 1, \ldots, N \), which can be thought of as a realization of a random variable \( X \), whose expected value \( \mu = \mathbb{E}[X] \) we want to estimate. Basic inferential statistics suggests using the sample mean \( \bar{X} \) and sample standard deviation \( S \) to build a confidence interval with confidence level \( 1 - \alpha \):

\[
\bar{X} \pm z_{1-\alpha/2} \frac{S}{\sqrt{N}},
\]

where we assume that the sample size \( N \) is large enough to warrant the use of quantiles \( z_{1-\alpha/2} \) from the standard normal distribution.\(^{15}\) This form of a confidence interval is so deceptively simple that we may forget the conditions for its validity.

#### Example 1.7 Estimating average waiting time

Consider the queuing system of Section 1.3.3.1, but now assume that there are multiple servers, possibly bank tellers. We might be interested in investigating the trade-off between the number of servers, which has a significant impact on cost, and the average waiting time in the queue, which is a measure of service quality. Using Monte Carlo, we may simulate a fairly large number \( N \) of customers and tally the waiting time \( W_k \), \( k = 1, \ldots, N \), for each of them. Based on this sample, we may estimate the average waiting time by the sample mean \( \bar{W} \). Now can we use Eq. (1.18) to build a confidence interval for the average waiting time?

By doing so, we would commit a few mistakes:

1. The confidence interval is exact for a normal population, i.e., when observations are normally distributed. Clearly, there is no reason to believe that waiting times are normally distributed.

2. What is the waiting time of the first customer entering the system? Clearly, this is zero, as the first customer is so lucky that she finds all of the servers idle. In such a system, we must account for

\(^{15}\)The quantile \( z_{1-\alpha/2} \) is a number such that \( \mathbb{P}(Z \leq z_{1-\alpha/2}) = 1 - \alpha/2 \), where \( Z \sim \mathcal{N}(0,1) \) is standard normal. Outlier analysis is treated in Chapter 7.
a transient phase, and we should collect statistics only when the system is in steady state.

3. Last but not least, the common way to estimate standard deviation assumes independence among the observations. However, successive waiting times are obviously positively correlated (the degree of correlation depends on the load on the system; for high utilization levels, the correlation is rather strong).

We may find a remedy to overcome these issues, based on a batching strategy, as we shall see in Chapter 7.

The last two mistakes underlined in Example 1.7 remind us that standard inferential statistics rely on a sequence of i.i.d. (independent and identically distributed) random variables. In nontrivial Monte Carlo methods this condition may not apply and due care must be exerted. Another issue that must not be disregarded is bias: Is it always the case that $E[\bar{X}] = \theta$, i.e., that the expected value of the sample mean we are taking is the parameter in which we are interested? This is certainly so if we are sampling a population and the parameter is actually the population mean $\mu$. In general, estimators can be biased.

Example 1.8 Bias in option pricing

Monte Carlo methods are commonly used to price options, i.e., financial derivatives written on an underlying asset. Consider, for instance, an American-style put option, written on a stock that pays no dividends during the option life. This option gives its holder the right to sell the underlying stock at the strike price $K$, up to option maturity $T$. Whenever the stock price $S(t)$ at time $t$, for $t \in [0, T]$, is smaller than $K$, the option is “in-the-money,” and the holder can exercise her right immediately, earning a payoff $K - S(t)$; to see why this is the payoff, consider that the holder can buy the stock share at the current (spot) price $S(t)$ and sell it immediately at the strike price $K > S(t)$. This is a somewhat idealized view, as it neglects transaction costs and the fact that market prices move continuously. However, for some derivatives that are settled in cash, like index options, this is really the payoff. However, when is it really optimal to exercise? Should we wait for better opportunities? Finding the answer requires the solution of a stochastic dynamic optimization problem, whereby we seek to maximize the payoff, whose output is an optimal exercise policy. The fair option value is related to the expected payoff resulting from the application of this optimal policy. When the problem is challenging,
we may settle for an approximate policy. Unfortunately, a suboptimal policy introduces a low bias, since we fall short of achieving the truly optimal payoff.

Leaving the aforementioned issues aside, the form of the confidence interval in Eq. (1.18) shows the role of estimator's variance:

\[
\text{Var}(X) = \frac{\sigma}{N},
\]

where \( \sigma \) is the (usually unknown) variance of each observation. This implies that the half-length of the confidence interval is given by \( z_{1-\alpha/2} \sigma / \sqrt{N} \), which is both good and bad news:

- The nice fact is that when the sample size \( N \) is increased, the confidence interval shrinks in a way that does not depend on problem dimensionality.\(^{16}\) This is why Monte Carlo methods are naturally suitable to high-dimensional problems.

- The not so nice fact is that this convergence is related to the square root of \( N \). To get the message loud and clear, this implies that to gain one order of precision, i.e., to divide the length of the confidence interval by a factor of 10, we should multiply \( N \) by a factor of 100.

The last observation explains why a huge number of replications may be needed in order to obtain reliable estimates. This brute force approach may be feasible thanks to another nice feature of Monte Carlo methods: They lend themselves to parallelization, and the availability of multicore machines at cheap prices is certainly helpful. However, a smarter approach does not rely on brute computational force. Using proper variance reduction techniques, discussed in Chapter 8, we may be able to reduce \( \sigma \) without introducing any bias in the estimator. Even this improvement may not be enough when dealing with complex multi-stage optimization problems, though.

1.5.2 PHILOSOPHICAL ISSUES

The issues that we have discussed in the previous section are technical in nature, and can be at least partially overcome by suitable tricks of the trade. However, there are some deeper issues that one must be aware of when tackling a problem by Monte Carlo, especially in risk management.\(^{17}\) While in other cases we might be interested in an average performance measure, in risk management

\(^{16}\)Well, we should say that there is no explicit dependence on dimensionality, even though this may affect variability, not to mention computational effort.

\(^{17}\)We shall deal with risk measurement and management in Chapter 13.
we are interested in the "bad tail" of a distribution, where extreme events may occur, such as huge losses in a financial portfolio.

Example 1.9 Value-at-risk

One of the most common (and most criticized) financial risk measures is value-at-risk (VaR). Technically, VaR is the quantile of the probability distribution of loss on a portfolio of assets, over a specified time horizon, with a given confidence level. In layman's terms, if the one-day VaR at 99% level is $1000, it means that we are "99% sure" that we will not lose more than $1000 the next day. If we know the probability density of loss, finding the quantile is a rather easy matter. In general, however, we may hold a portfolio with widely different asset classes, including stock shares, bonds, and perhaps quite sophisticated derivatives, whose value depends on a large number of underlying factors. Monte Carlo methods can be used to sample a suitable set of scenarios in terms of factors that are mapped to scenarios in terms of portfolio value; then, we may estimate the quantile in which we are interested. It stands to reason that estimating VaR at 99.9% will take a larger computational effort than estimating VaR at 90%, since we have to deal with extreme events. Using brute force, or clever variance reduction, we might be able to tackle such a task. However, there are more serious difficulties:

- Can we really trust our probabilistic model when we deal with very rare, but potentially disastrous events?
- What about the reliability of the estimates of small probabilities?
- What happens to correlations under extreme conditions?
- Did we include every risk factors in our model? Assuming we did so in financial terms, what about the consequence of an earthquake?

Sometimes, a line is drawn between uncertainty and risk. The latter term is reserved to cases in which the rules of the game are clear and perfectly known, as in the case of dice throwing and coin flipping. Uncertainty, in this framework, is a more general concept that is not really reduced to the uncertainty in the realization of random variables. To get the picture, let us consider the scenario tree depicted in Fig. 1.15. The tree can be interpreted in terms of a discrete probability distribution: Future scenarios \( S_k \) are associated with probabilities \( \pi_k \). Risk has to do with the case in which both possible scenarios and their probabilities are known. However, we may have another level of uncertainty when the probabilities are not really known, i.e., we have uncertainty about the uncertainty. Indeed, when dealing with extreme risks, the very notion of
probability may be questioned: Are we talking about frequentist probabilities, or subjective/Bayesian views? We may even fail to consider all of the scenarios. The black scenario in the figure is something that we have failed to consider among the plausible contingencies. The term black swan is commonly used, as well as unk-unks, i.e., unknown unknowns. To add some complexity to the picture, in most stochastic models uncertainty is exogenous. In social systems, however, uncertainty is at least partially endogenous. To see concrete examples, think of the effect of herding behavior in finance and trading in thin and illiquid markets.

To see the issue within a more general setting, imagine a mathematical model of a physical system described by a difficult partial differential equation. Can we obtain brand new knowledge by solving the equation using accurate numerical methods? Arguably, the knowledge itself is encoded in the equation. Solving it may be useful in understanding the consequence of that knowledge, possibly invalidating the model when results do not match observed reality. By a similar token, Monte Carlo methods are numerical methods applied to a stochastic model. There is high value in flexible and powerful methods that allow tackling complex models that defy analytical solution. But if the very assumptions of the model are flawed, the results will be flawed as well. If we analyze risks using sophisticated Monte Carlo strategies, we may fall into the trap of a false sense of security, and pay the consequence if some relevant source of uncertainty has not been included in the model or if probabilities have been poorly assessed. Monte Carlo is an excellent tool, but it is just that: a tool.

1.6 Software tools for Monte Carlo simulation

In general, Monte Carlo simulations require the following:

1. The possibility of describing the system behavior using an imperative programming language or a connection of graphical blocks
2. Random number generators
3. Statistical tools to collect and analyze results
4. Visualization/animation facilities

There is a wide array of software systems that can be used to meet the above requirements, which may be more or less relevant, depending on the underlying kind of model:

- General-purpose programming languages like C++, Visual Basic, or Java.
- Spreadsheet add-ins like @Risk\(^{18}\)
- Software environments like R and MATLAB,\(^{19}\) which also allow for classical imperative programming, in a relatively high-level language
- Graphical editors for the description of a system in terms of interconnected building blocks like Simulink, which is an outgrowth of MATLAB, and Arena\(^{20}\)

All of these options have advantages and disadvantages in terms of cost, ease of use, flexibility, readability, and ease of maintenance of the models. Clearly, the maximum power is associated with general-purpose languages. By using proper object-oriented methods, we may also build reusable libraries, and compiled general-purpose languages are definitely the best option in terms of execution speed. However, they require a non-negligible coding effort, particularly if some essential modelling infrastructure must be built from scratch. Furthermore, with respect to more powerful programming languages like MATLAB, they require many more lines of code. This is due to the need of declaring variables, which allows for good type checking and efficient compiled code, but it is definitely bad for rapid prototyping. Many lines of code may also obscure the underlying model. Furthermore, visualization and random number generation require a significant coding effort as well, unless one relies on external libraries.

Spreadsheets provide the user with a familiar interface, and this can be exploited both by writing code, e.g., in VBA for Microsoft Excel, or by using add-ins, which automate some tasks and typically provide the user with much better quality random number generators. However, if VBA code is written, we essentially fall back on general-purpose programming languages; if a ready-to-use tool is adopted, we may lack flexibility.

On the opposite side of the spectrum we find tools relying on graphical editors to connect building blocks. This is really essential when dealing with complex discrete-event systems. When modeling manufacturing systems, building everything from scratch is a daunting and error-prone effort. A similar consideration applies when simulating nonlinear systems with complicated dynamics. Indeed, this is a domain where such tools like Arena or Simulink are quite help-

\(^{18}\) See http://www.palisade.com and the copyright notices there.
\(^{19}\) See http://www.mathworks.com and the copyright notices there.
\(^{20}\) See http://www.arenasimulation.com and the copyright notices there.
ful. Unfortunately, these tools are not really aimed at problems in finance and economics, and are rather expensive.

The kind of simulation models we are concerned with in this book are somewhere in between, as they do not involve overly complex event bookkeeping, but may require careful discretization of stochastic differential equations. They do not involve the level of sophistication needed to simulate complicate engineering systems featuring non-linearities, but we do face the additional issues related with uncertainty. Hence, we need a set of functions enabling us to sample a variety of probability distributions, as well as to statistically analyze the output. Therefore, in most financial and some economic applications, a good compromise between flexibility and power is represented by numerical and statistical computing environments like MATLAB and R. They offer a library of statistical functions, a rich programming language, and powerful visualization tools. They are both excellent tools for teaching, and even if they are not best suited in terms of efficiency, they can be used to validate a model and to prototype code; the code can be then translated to a more efficient compiled language. In this book, we will adopt R, which has the very welcome feature of being free.\footnote{I have made my point about my personal preference between MATLAB and R in the Preface, and I will not repeat myself here. For more information on the use of MATLAB for financial applications, see http://www.mathworks.com/financial-services/, and for an illustration of using MATLAB in Monte Carlo financial applications see [2].} All the code displayed in this book is available on my website:

http://staff.polito.it/paolo.brandimarte

1.7 Prerequisites

The prerequisites for reading this book involve:

- Mathematical background
- Financial/economical background
- Technical/programming background

Generally speaking, the requirements are rather mild and any professional or advanced undergraduate student in mathematics, statistics, economics, finance, and engineering should be able to take advantage of this book.

1.7.1 MATHEMATICAL BACKGROUND

An obvious prerequisite for a book on Monte Carlo methods is represented by the basics of probability theory and inferential statistics, including:

- Events and conditional probabilities
• Discrete and continuous random variables (including distribution functions, quantiles, moments)
• A bit of stochastic processes
• A modicum of multivariate distributions, with emphasis on concepts related to covariance and correlation
• Statistical inference (confidence intervals and linear regression)

All of the rest will be explained when needed, including some basic concepts about stochastic integration and stochastic differential equations, as well as copula theory. Note that we will never use measure-theoretic probability. In the chapters involving optimization, we also assume some familiarity with linear and nonlinear programming. The required background is typically covered in undergraduate courses, and any basic book could be used for a brush-up. In my book on quantitative methods [3], all of the required background is given, both for probability/statistics and optimization applications.

1.7.2 FINANCIAL BACKGROUND

This book does touch some topics that may be useful for an economist, but the main focus is on finance. To keep this book to a reasonable size, I will not provide much background on, say, portfolio management and option pricing theory. When illustrating examples, we will just state which kind of mathematical problem we have to face, then we will proceed and solve it. To be specific, when illustrating Monte Carlo methods for option pricing, I will just recapitulate what options are, but I will not provide any deep explanation of risk-neutral pricing principles; a quick outline is provided in Section 3.9. The reader interested in that background could consult, e.g., [1] or [10].

1.7.3 TECHNICAL BACKGROUND

I assume that the reader has a working R installation and knows how to run a simple session, and how to source functions and scripts. There are plenty of online tutorials addressing the needs of a newcomer. Therefore, rather than adding a dry appendix quickly reviewing R programming, I will provide R notes along the way, when discussing concrete examples. Certainly, some familiarity with the basic concepts of any programming language is essential. However, we will not need complicated data structures, and we will also do without sophisticated object-oriented principles. The code in this book is provided only to help the reader in understanding Monte Carlo methods, but it is far from being industrial strength, reusable, and efficient code.
For further reading

- In this book, we will deal with Monte Carlo methods at an intermediate level. Complementary references, including some more advanced material, are [7, 13, 15]. In particular, [13] is a comprehensive treatment, offering pieces of MATLAB code. See also [18] for a treatment with emphasis on Bayesian statistics. By the way, a quite readable book on the important topic of Bayesian statistics, including Monte Carlo methods in R, is [14].

- The role of Monte Carlo methods for optimization is discussed in [8] and [19]; see [17] for an extensive treatment of dynamic stochastic optimization based on Monte Carlo simulation.

- Stochastic programming theory and models are treated, e.g., in [11].

- Probably, most readers of this book are interested in financial engineering. An excellent treatment of Monte Carlo methods for financial engineering is given in [9]; see also [12] or, at a more elementary level, [2], which is MATLAB based.

- On the technical side, good readings on R programming are [6] and [16].

References


