Monte Carlo Methods

Dirk P. Kroese

Department of Mathematics
School of Mathematics and Physics
The University of Queensland

kroese@maths.uq.edu.au
http://www.maths.uq.edu.au/~kroese

Preface

Many numerical problems in science, engineering, finance, and statistics are solved nowadays through **Monte Carlo methods**; that is, through random experiments on a computer. The purpose of this AMSI Summer School course is to provide a comprehensive introduction to Monte Carlo methods, with a mix of theory, algorithms (pseudo + actual), and applications.

These notes present a highly condensed version of:

D.P. Kroese, T. Taimre, Z.I. Botev. *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics, John Wiley & Sons, New York, 2011.

See also the Handbook's website: `www.montecarlohandbook.org`.

Since the Handbook is over 772 pages thick, with 21 chapters, I had to heavily cut back the contents of the Handbook to a size that is manageable to teach within one semester. I have tried to make these notes fairly self-contained, while retaining the general flavour of the Handbook. However, it was not always possible to keep the logical connections between chapters in the Handbook. For an advanced understanding of some material, including bibliographic references, it will be necessary to consult the corresponding passages in the Handbook.

Brisbane, 2011

Dirk Kroese

Contents

1 Uniform Random Number Generation **9**

1.1 Random Numbers 9

1.1.1 Properties of a Good Random Number Generator 10

1.1.2 Choosing a Good Random Number Generator 11

1.2 Generators Based on Linear Recurrences 12

1.2.1 Linear Congruential Generators 12

1.2.2 Multiple-Recursive Generators 13

1.2.3 Matrix Congruential Generators 13

1.2.4 Modulo 2 Linear Generators 14

1.3 Combined Generators 16

1.4 Tests for Random Number Generators 17

1.4.1 Equidistribution (or Frequency) Tests 20

1.4.2 Serial Tests 21

1.4.3 Gap Tests 21

1.4.4 Poker or Partition Tests 21

1.4.5 Coupon Collector's Tests 22

1.4.6 Permutation Tests 22

1.4.7 Run Tests 22

1.4.8 Maximum-of-d Tests 22

1.4.9 Collision Tests 23

1.4.10 Rank of Binary Matrix Tests 23

1.4.11 Birthday Spacings Tests 23

1.5 Exercises 24

2 Random Variable Generation **25**

2.1 Generic Algorithms Based on Common Transformations 25

2.1.1 Inverse-Transform Method 26

2.1.2 Other Transformation Methods 28

2.1.3 Table Lookup Method 34

2.1.4 Alias Method 35

2.1.5 Acceptance–Rejection Method 38

2.2 Generation Methods for Multivariate Random Variables 39

2.3 Generating Random Vectors Uniformly Distributed in a Unit Hyperball and Hypersphere 40

2.4 Generating Random Permutations 41

2.5 Exercises 42

3 Probability Distributions **45**
3.1 Discrete Distributions . . . 45
3.1.1 Bernoulli Distribution . . . 45
3.1.2 Binomial Distribution . . . 46
3.1.3 Geometric Distribution . . . 46
3.1.4 Poisson Distribution . . . 47
3.1.5 Uniform Distribution (Discrete Case) . . . 48
3.2 Continuous Distributions . . . 49
3.2.1 Beta Distribution . . . 49
3.2.2 Cauchy Distribution . . . 50
3.2.3 Exponential Distribution . . . 51
3.2.4 Gamma Distribution . . . 51
3.2.5 Normal Distribution . . . 53
3.2.6 Uniform Distribution (Continuous Case) . . . 54
3.3 Multivariate Distributions . . . 55
3.3.1 Dirichlet Distribution . . . 55
3.3.2 Multivariate Normal Distribution . . . 56
3.3.3 Multivariate Student's t Distribution . . . 58
3.4 Exercises . . . 59

4 Random Process Generation **61**
4.1 Gaussian Processes . . . 61
4.1.1 Markovian Gaussian Processes . . . 62
4.2 Markov Chains . . . 63
4.3 Markov Jump Processes . . . 66
4.4 Poisson Processes . . . 69
4.5 Wiener Process and Brownian Motion . . . 73
4.6 Stochastic Differential Equations and Diffusion Processes . . . 75
4.6.1 Euler's Method . . . 76
4.7 Brownian Bridge . . . 78
4.8 Geometric Brownian Motion . . . 80
4.9 Ornstein–Uhlenbeck Process . . . 82
4.10 Exercises . . . 84

5 Markov Chain Monte Carlo **87**
5.1 Metropolis–Hastings Algorithm . . . 87
5.1.1 Independence Sampler . . . 88
5.1.2 Random Walk Sampler . . . 89
5.2 Gibbs Sampler . . . 91
5.3 Hit-and-Run Sampler . . . 95
5.4 Exercises . . . 100

6 Variance Reduction **103**
6.1 Variance Reduction Example . . . 103
6.2 Antithetic Random Variables . . . 105
6.3 Control Variables . . . 108
6.4 Conditional Monte Carlo . . . 110

6.5 Importance Sampling . . . 113
6.5.1 Minimum-Variance Density . . . 114
6.5.2 Variance Minimization Method . . . 115
6.5.3 Cross-Entropy Method . . . 117
6.6 Exercises . . . 119

7 Estimation of Derivatives 123
7.1 Gradient Estimation . . . 123
7.2 Finite Difference Method . . . 125
7.3 Infinitesimal Perturbation Analysis . . . 128
7.4 Score Function Method . . . 129
7.4.1 Score Function Method With Importance Sampling . . . 132

8 Randomized Optimization 137
8.1 Stochastic Approximation . . . 137
8.2 Stochastic Counterpart Method . . . 142
8.3 Simulated Annealing . . . 145
8.4 Evolutionary Algorithms . . . 148
8.4.1 Genetic Algorithms . . . 149
8.4.2 Differential Evolution . . . 150
8.5 Exercises . . . 153

9 Cross-Entropy Method 155
9.1 Cross-Entropy Method . . . 155
9.2 Cross-Entropy Method for Estimation . . . 156
9.3 Cross-Entropy Method for Optimization . . . 159
9.3.1 Combinatorial Optimization . . . 161
9.3.2 Continuous Optimization . . . 163
9.3.3 Constrained Optimization . . . 165
9.3.4 Noisy Optimization . . . 168
9.4 Exercises . . . 169

Chapter 1

Uniform Random Number Generation

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.

John von Neumann

This chapter gives an introduction of techniques and algorithms for generating uniform random numbers. Various empirical tests for randomness are also provided.

1.1 Random Numbers

At the heart of any Monte Carlo method is a **random number generator**: a procedure that produces an infinite stream

$$U_1, U_2, U_3, \ldots \overset{\text{iid}}{\sim} \mathsf{Dist}$$

of random variables that are independent and identically distributed (iid) according to some probability distribution Dist. When this distribution is the uniform distribution on the interval (0,1) (that is, $\mathsf{Dist} = \mathsf{U}(0,1)$), the generator is said to be a **uniform random number generator**. Most computer languages already contain a built-in uniform random number generator. The user is typically requested only to input an initial number, called the **seed**, and upon invocation the random number generator produces a sequence of independent uniform random variables on the interval $(0,1)$. In MATLAB, for example, this is provided by the `rand` function.

The concept of an infinite iid sequence of random variables is a mathematical abstraction that may be impossible to implement on a computer. The best one can hope to achieve in practice is to produce a sequence of "random" numbers with statistical properties that are indistinguishable from those of a true sequence of iid random variables. Although physical generation methods based on universal background radiation or quantum mechanics seem to offer a stable source of such true randomness, the vast majority of current random number

generators are based on simple algorithms that can be easily implemented on a computer. Such algorithms can usually be represented as a tuple $(\mathcal{S}, f, \mu, \mathcal{U}, g)$, where

- $\mathcal{S}$ is a finite set of **states**,
- f is a function from $\mathcal{S}$ to $\mathcal{S}$,
- μ is a probability distribution on $\mathcal{S}$,
- $\mathcal{U}$ is the **output space**; for a uniform random number generator $\mathcal{U}$ is the interval $(0, 1)$, *and we will assume so from now on, unless otherwise specified*,
- g is a function from $\mathcal{S}$ to $\mathcal{U}$.

A random number generator then has the following structure:

Algorithm 1.1 (Generic Random Number Generator)

1. **Initialize***: Draw the seed S_0 from the distribution μ on $\mathcal{S}$. Set $t = 1$.*
2. **Transition***: Set $S_t = f(S_{t-1})$.*
3. **Output***: Set $U_t = g(S_t)$.*
4. **Repeat***: Set $t = t + 1$ and return to Step 2.*

The algorithm produces a sequence $U_1, U_2, U_3, \ldots$ of **pseudorandom numbers** — we will refer to them simply as **random numbers**. Starting from a certain seed, the sequence of states (and hence of random numbers) must repeat itself, because the state space is finite. The smallest number of steps taken before entering a previously visited state is called the **period length** of the random number generator.

1.1.1 Properties of a Good Random Number Generator

What constitutes a good random number generator depends on many factors. It is always advisable to have a variety of random number generators available, as different applications may require different properties of the random generator. Below are some desirable, or indeed essential, properties of a good uniform random number generator:

1. *Pass statistical tests*: The ultimate goal is that the generator should produce a stream of uniform random numbers that is indistinguishable from a genuine uniform iid sequence. Although from a theoretical point of view this criterion is too imprecise and even infeasible, from a practical point of view this means that the generator should pass a battery of simple statistical tests designed to detect deviations from uniformity and independence. We discuss such tests in Section 1.4.
2. *Theoretical support*: A good generator should be based on sound mathematical principles, allowing for a rigorous analysis of essential properties of the generator. Examples are linear congruential generators and multiple-recursive generators discussed in Sections 1.2.1 and 1.2.2.

3. *Reproducible*: An important property is that the stream of random numbers is reproducible without having to store the complete stream in memory. This is essential for testing and variance reduction techniques. Physical generation methods cannot be repeated unless the entire stream is recorded.
4. *Fast and efficient*: The generator should produce random numbers in a fast and efficient manner, and require little storage in computer memory. Many Monte Carlo techniques for optimization and estimation require billions or more random numbers. Current physical generation methods are no match for simple algorithmic generators in terms of speed.
5. *Large period*: The period of a random number generator should be extremely large — on the order of 10^{50} — in order to avoid problems with duplication and dependence. Most early algorithmic random number generators were fundamentally inadequate in this respect.
6. *Multiple streams*: In many applications it is necessary to run multiple independent random streams in parallel. A good random number generator should have easy provisions for multiple independent streams.
7. *Cheap and easy*: A good random number generator should be cheap and not require expensive external equipment. In addition, it should be easy to install, implement, and run. In general such a random number generator is also more easily portable over different computer platforms and architectures.
8. *Not produce 0 or 1*: A desirable property of a random number generator is that both 0 and 1 are excluded from the sequence of random numbers. This is to avoid division by 0 or other numerical complications.

1.1.2 Choosing a Good Random Number Generator

Choosing a good random generator is like choosing a new car: for some people or applications speed is preferred, while for others robustness and reliability are more important. For Monte Carlo simulation the distributional properties of random generators are paramount, whereas in coding and cryptography unpredictability is crucial.

Nevertheless, as with cars, there are many poorly designed and outdated models available that should be avoided. Indeed several of the standard generators that come with popular programming languages and computing packages can be appallingly poor.

Two classes of generators that have overall good performance are:

1. *Combined multiple recursive generators*, some of which have excellent statistical properties, are simple, have large period, support multiple streams, and are relatively fast. A popular choice is L'Ecuyer's `MRG32k3a` (see Section 1.3), which has been implemented as one of the core generators in MATLAB (from version 7), VSL, SAS, and the simulation packages SSJ, Arena, and Automod.

2. *Twisted general feedback shift register generators*, some of which have very good equidistributional properties, are among the fastest generators available (due to their essentially binary implementation), and can have extremely long periods. A popular choice is Matsumoto and Nishimura's Mersenne twister `MT19937ar` (see Section 1.2.4), which is currently the default generator in MATLAB.

In general, a good uniform number generator has *overall* good performance, in terms of the criteria mentioned above, but is not usually the top performer over all these criteria. In choosing an appropriate generator it pays to remember the following.

- Faster generators are not necessarily better (indeed, often the contrary is true).
- A small period is in general bad, but a larger period is not necessarily better.
- Good equidistribution is a necessary requirement for a good generator but not a sufficient requirement.

1.2 Generators Based on Linear Recurrences

The most common methods for generating pseudorandom sequences use simple linear recurrence relations.

1.2.1 Linear Congruential Generators

A **linear congruential generator** (LCG) is a random number generator of the form of Algorithm 1.1, with state $S_t = X_t \in \{0, \ldots, m-1\}$ for some strictly positive integer m called the **modulus**, and state transitions

$$X_t = (aX_{t-1} + c) \bmod m \;, \quad t = 1, 2, \ldots \;, \tag{1.1}$$

where the **multiplier** a and the **increment** c are integers. Applying the modulo-m operator in (1.1) means that $aX_{t-1} + c$ is divided by m, and the remainder is taken as the value for X_t. Note that the multiplier and increment may be chosen in the set $\{0, \ldots, m-1\}$. When $c = 0$, the generator is sometimes called a **multiplicative congruential generator**. Most existing implementations of LCGs are of this form — in general the increment does not have a large impact on the quality of an LCG. The output function for an LCG is simply

$$U_t = \frac{X_t}{m} \;.$$

Example 1.1 (Minimal Standard LCG) An often-cited LCG is that of Lewis, Goodman, and Miller, who proposed the choice $a = 7^5 = 16807$, $c = 0$, and $m = 2^{31} - 1 = 2147483647$. This LCG passes many of the standard statistical tests and has been successfully used in many applications. For this

reason it is sometimes viewed as the *minimal standard* LCG, against which other generators should be judged.

Although the generator has good properties, its period $(2^{31} - 2)$ and statistical properties no longer meet the requirements of modern Monte Carlo applications.

A comprehensive list of classical LCGs and their properties can be found on Karl Entacher's website:

`http://random.mat.sbg.ac.at/results/karl/server/`

1.2.2 Multiple-Recursive Generators

A **multiple-recursive generator** (MRG) of **order** k, is a random number generator of the form of Algorithm 1.1, with state $S_t = \mathbf{X}_t = (X_{t-k+1}, \ldots, X_t)^\top \in \{0, \ldots, m-1\}^k$ for some modulus m and state transitions defined by

$$X_t = (a_1 X_{t-1} + \cdots + a_k X_{t-k}) \bmod m \ , \quad t = k, k+1, \ldots \ , \tag{1.2}$$

where the **multipliers** $\{a_i, i = 1, \ldots, k\}$ lie in the set $\{0, \ldots, m-1\}$. The output function is often taken as

$$U_t = \frac{X_t}{m} \ .$$

The maximum period length for this generator is $m^k - 1$, which is obtained if (a) m is a prime number and (b) the polynomial $p(z) = z^k - \sum_{i=1}^{k-1} a_i z^{k-i}$ is *primitive* using modulo m arithmetic. To yield fast algorithms, all but a few of the $\{a_i\}$ should be 0.

MRGs with very large periods can be implemented efficiently by combining several smaller-period MRGs (see Section 1.3).

1.2.3 Matrix Congruential Generators

An MRG can be interpreted and implemented as a **matrix multiplicative congruential generator**, which is a random number generator of the form of Algorithm 1.1, with state $S_t = \mathbf{X}_t \in \{0, \ldots, m-1\}^k$ for some modulus m, and state transitions

$$\mathbf{X}_t = (A\mathbf{X}_{t-1}) \bmod m, \quad t = 1, 2, \ldots \ , \tag{1.3}$$

where A is an invertible $k \times k$ matrix and $\mathbf{X}_t$ is a $k \times 1$ vector. The output function is often taken as

$$\mathbf{U}_t = \frac{\mathbf{X}_t}{m} \ , \tag{1.4}$$

yielding a vector of uniform numbers in $(0, 1)$. Hence, here the output space $\mathcal{U}$ for the algorithm is $(0, 1)^k$. For fast random number generation, the matrix A should be sparse.

To see that the multiple-recursive generator is a special case, take

$$A = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix} \quad \text{and} \quad \mathbf{X}_t = \begin{pmatrix} X_t \\ X_{t+1} \\ \vdots \\ X_{t+k-1} \end{pmatrix} . \tag{1.5}$$

Obviously, the matrix multiplicative congruential generator is the k-dimensional generalization of the multiplicative congruential generator. A similar generalization of the multiplicative recursive generator — replacing the multipliers $\{a_i\}$ with matrices, and the scalars $\{X_t\}$ with vectors in (1.2) — yields the class of **matrix multiplicative recursive generators**.

1.2.4 Modulo 2 Linear Generators

Good random generators must have very large state spaces. For an LCG this means that the modulus m must be a large integer. However, for multiple recursive and matrix generators it is not necessary to take a large modulus, as the state space can be as large as m^k. Because binary operations are in general faster than floating point operations (which are in turn faster than integer operations), it makes sense to consider random number generators that are based on linear recurrences modulo 2. A general framework for such random number generators is to map k-bit state vector $\mathbf{X}_t = (X_{t,1}, \dots, X_{t,k})^\top$ via a linear transformation to a w-bit output vector $\mathbf{Y}_t = (Y_{t,1}, \dots, Y_{t,w})^\top$, from which the random number $U_t \in (0, 1)$ is obtained by *bitwise decimation*. More precisely, the procedure is as follows.

Algorithm 1.2 (Generic Linear Recurrence Modulo 2 Generator)

1. **Initialize***: Draw the seed* $\mathbf{X}_0$ *from the distribution* μ *on the state space* $\mathcal{S} = \{0, 1\}^k$*. Set* $t = 1$*.*

2. **Transition***: Set* $\mathbf{X}_t = A\mathbf{X}_{t-1}$*.*

3. **Output***: Set* $\mathbf{Y}_t = B\mathbf{X}_t$ *and return*

$$U_t = \sum_{\ell=1}^{w} Y_{t,\ell}\, 2^{-\ell} .$$

4. **Repeat***: Set* $t = t + 1$ *and return to Step 2.*

Here, A and B are $k \times k$ and $w \times k$ binary matrices, respectively, and all operations are performed modulo 2. In algebraic language, the operations are performed over the finite field $\mathbb{F}_2$, where addition corresponds to the bitwise `XOR` operation (in particular, $1 + 1 = 0$). The integer w can be thought of as the word length of the computer (that is, $w = 32$ or 64). Usually k is taken much larger than w.

Example 1.2 (Linear Feedback Shift Register Generator) The **Tausworthe** or **linear feedback shift register** (LFSR) generator is an MRG of the form (1.2) with $m = 2$, but with output function

$$U_t = \sum_{\ell=1}^{w} X_{ts+\ell-1}\, 2^{-\ell} ,$$

for some $w \leqslant k$ and $s \geqslant 1$ (often one takes $s = w$). Thus, a binary sequence $X_0, X_1, \ldots$ is generated according to the recurrence (1.2), and the t-th "word" $(X_{ts}, \ldots, X_{ts+w-1})^\top$, $t = 0, 1, \ldots$ is interpreted as the binary representation of the t-th random number.

This generator can be put in the framework of Algorithm 1.2. Namely, the state at iteration t is given by the vector $\mathbf{X}_t = (X_{ts}, \ldots, X_{ts+k-1})^\top$, and the state is updated by advancing the recursion (1.2) over s time steps. As a result, the transition matrix A in Algorithm 1.2 is equal to the s-th power of the "1-step" transition matrix given in (1.5). The output vector $\mathbf{Y}_t$ is obtained by simply taking the first w bits of $\mathbf{X}_t$; hence $B = [I_w \;\; O_{w\times(k-w)}]$, where I_w is the identity matrix of dimension w and $O_{w\times(k-w)}$ the $w \times (k-w)$ matrix of zeros.

For fast generation most of the multipliers $\{a_i\}$ are 0; in many cases there is often only *one* other non-zero multiplier a_r apart from a_k, in which case

$$X_t = X_{t-r} \oplus X_{t-k} , \tag{1.6}$$

where $\oplus$ signifies addition modulo 2. The same recurrence holds for the states (vectors of bits); that is,

$$\mathbf{X}_t = \mathbf{X}_{t-r} \oplus \mathbf{X}_{t-k} ,$$

where addition is defined componentwise.

The LFSR algorithm derives it name from the fact that it can be implemented very efficiently on a computer via **feedback shift registers** — binary arrays that allow fast shifting of bits.

Generalizations of the LFSR generator that all fit the framework of Algorithm 1.2 include the **generalized feedback shift register** generators and the **twisted** versions thereof, the most popular of which are the **Mersenne twisters**. A particular instance of the Mersenne twister, `MT19937`, has become widespread, and has been implemented in software packages such as SPSS and MATLAB. It has a huge period length of $2^{19937}-1$, is very fast, has good equidistributional properties, and passes most statistical tests. The latest version of the code may be found at

`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`

Two drawbacks are that the initialization procedure and indeed the implementation itself is not straightforward. Another potential problem is that the algorithm recovers too slowly from the states near zero. More precisely, after a state with very few 1s is hit, it may take a long time (several hundred thousand steps) before getting back to some state with a more equal division between 0s and 1s.

1.3 Combined Generators

A significant leap forward in the development of random number generators was made with the introduction of **combined generators**. Here the output of several generators, which individually may be of poor quality, is combined, for example by shuffling, adding, and/or selecting, to make a superior quality generator.

Example 1.3 (Wichman–Hill) One of the earliest combined generators is the Wichman–Hill generator, which combines three LCGs:

$$\begin{aligned} X_t &= (171\, X_{t-1}) \bmod m_1 \qquad (m_1 = 30269)\ , \\ Y_t &= (172\, Y_{t-1}) \bmod m_2 \qquad (m_2 = 30307)\ , \\ Z_t &= (170\, Z_{t-1}) \bmod m_3 \qquad (m_3 = 30323)\ . \end{aligned}$$

These random integers are then combined into a single random number

$$U_t = \frac{X_t}{m_1} + \frac{Y_t}{m_2} + \frac{Z_t}{m_3} \mod 1\ .$$

The period of the sequence of triples (X_t, Y_t, Z_t) is shown to be $(m_1 - 1)(m_2 - 1)(m_3 - 1)/4 \approx 6.95 \times 10^{12}$, which is much larger than the individual periods. Zeisel shows that the generator is in fact equivalent (produces the same output) as a multiplicative congruential generator with modulus $m = 27817185604309$ and multiplier $a = 16555425264690$.

The Wichman–Hill algorithm performs quite well in simple statistical tests, but since its period is not sufficiently large, it fails various of the more sophisticated tests, and is no longer suitable for high-performance Monte Carlo applications.

One class of combined generators that has been extensively studied is that of the **combined multiple-recursive generators**, where a small number of MRGs are combined. This class of generators can be analyzed theoretically in the same way as single MRG: under appropriate initialization the output stream of random numbers of a combined MRG is exactly the same as that of some larger-period MRG. Hence, to assess the quality of the generator one can employ the same well-understood theoretical analysis of MRGs. As a result, the multipliers and moduli in the combined MRG can be searched and chosen in a systematic and principled manner, leading to random number generators with excellent statistical properties. An important added bonus is that such algorithms lead to easy multi-stream generators.

L'Ecuyer has conducted an extensive numerical search and detailed theoretical analysis to find good combined MRGs. One of the combined MRGs that stood out was `MRG32k3a`, which employs two MRGs of order 3,

$$X_t = (1403580\, X_{t-2} - 810728\, X_{t-3}) \bmod m_1 \quad (m_1 = 2^{32} - 209 = 4294967087)\ ,$$

$$Y_t = (527612\, Y_{t-1} - 1370589\, Y_{t-3}) \bmod m_2 \quad (m_2 = 2^{32} - 22853 = 4294944443)\ ,$$

and whose output is

$$U_t = \begin{cases} \dfrac{X_t - Y_t + m_1}{m_1 + 1} & \text{if } X_t \leqslant Y_t\,, \\ \dfrac{X_t - Y_t}{m_1 + 1} & \text{if } X_t > Y_t\,. \end{cases}$$

The period length is approximately 3×10^{57}. The generator `MRG32k3a` passes all statistical tests in today's most comprehensive test suit *TestU01* (see also Section 1.4) and has been implemented in many software packages, including MATLAB, Mathematica, Intel's MKL Library, SAS, VSL, Arena, and Automod. It is also the core generator in L'Ecuyer's SSJ simulation package, and is easily extendable to generate multiple random streams. An implementation in MATLAB is given below.

```
%MRG32k3a.m
m1=2^32-209; m2=2^32-22853;
ax2p=1403580; ax3n=810728;
ay1p=527612; ay3n=1370589;

X=[12345 12345 12345]; % Initial X
Y=[12345 12345 12345]; % Initial Y

N=100; % Compute the sequence for N steps
U=zeros(1,N);
for t=1:N
   Xt=mod(ax2p*X(2)-ax3n*X(3),m1);
   Yt=mod(ay1p*Y(1)-ay3n*Y(3),m2);
   if Xt <= Yt
       U(t)=(Xt - Yt + m1)/(m1+1);
   else
       U(t)=(Xt - Yt)/(m1+1);
   end
   X(2:3)=X(1:2); X(1)=Xt; Y(2:3)=Y(1:2); Y(1)=Yt;
end
```

Different *types* of generators can also be combined. For example, Marsaglia's `KISS99` (keep it simple stupid) generator combines two shift register generators with an LCG.

1.4 Tests for Random Number Generators

The quality of random number generators can be assessed in two ways. The first is to investigate the theoretical properties of the random number generator. Such properties include the period length of the generator and various measures

of uniformity and independence. This type of random number generator testing is called **theoretical**, as it does not require the actual output of the generator but only its algorithmic structure and parameters. Powerful theoretical tests are only feasible if the generators have a sufficiently simple structure, such as those of linear congruential and multiple-recursive methods and combined versions thereof.

A second type of test involves the application of a battery of statistical tests to the output of the generator, with the objective to detect deviations from uniformity and independence. Such type of tests are said to be **empirical**. In this course we consider only empirical tests. The ultimate goal remains to find uniform random number generators whose output is statistically indistinguishable (within reasonable computational time) from a sequence of iid uniform random variables. Hence, any candidate generator should pass a wide range of statistical tests that examine uniformity and independence. The general structure of such tests is often of the following form.

Algorithm 1.3 (Two-Stage Empirical Test for Randomness) *Suppose that* $\mathbf{U} = \{U_i\}$ *represents the output stream of the uniform random generator. Let* H_0 *be the hypothesis that the* $\{U_i\}$ *are iid from a* $\mathsf{U}(0,1)$ *distribution. Let* Z *be some deterministic function of* $\mathbf{U}$*.*

1. *Generate* N *independent copies* $Z_1, \ldots, Z_N$ *of* Z *and evaluate a test statistic* $T = T(Z_1, \ldots, Z_N)$ *for testing* H_0 *versus the alternative that* H_0 *is not true. Suppose that under* H_0 *the test statistic* T *has distribution or asymptotic (for large* N*) distribution* Dist_0*.*

2. *Generate* K *independent copies* $T_1, \ldots, T_K$ *of* T *and perform a goodness of fit test to test the hypothesis that the* $\{T_i\}$ *are iid from* Dist_0*.*

Such a test procedure is called a **two-stage** or **second-order** statistical test. The first stage corresponds to an ordinary statistical test, such as a χ^2 goodness of fit test, and the second stage combines K such tests by means of another goodness of fit test, such as the Kolmogorov–Smirnov or Anderson–Darling test. The following example demonstrates the procedure.

Example 1.4 (Binary Rank Test for the `drand48` Generator) The default random number generator in the C library is `drand48`, which implements an LCG with $a = 25214903917$, $m = 2^{48}$, and $c = 11$. We wish to examine if the output stream of this generator passes the *binary rank test* described in Section 1.4.10. For this test, the sequence $U_1, U_2, \ldots$ is first transformed to a binary sequence $B_1, B_2, \ldots$, for example, by taking $B_i = \mathrm{I}_{\{U_i \leqslant 1/2\}}$, and then the $\{B_i\}$ are arranged in a binary array, say with 32 rows and 32 columns. The first row of the matrix is $B_1, \ldots, B_{32}$, the second row is $B_{33}, \ldots B_{64}$, etc. Under H_0 the distribution of the rank (in modulo 2 arithmetic) R of this random matrix is given in (1.9). We generate $N = 200$ copies of R, and divide these into three classes: $R \leqslant 30$, $R = 31$, and $R = 32$. The expected number of ranks in these classes is by (1.9) equal to $E_1 = 200 \times 0.1336357$, $E_2 = 200 \times 0.5775762$, and

$E_3 = 200 \times 0.2887881$. This is compared with the observed number of ranks O_1, O_2, and O_3, via the χ^2 goodness of fit statistic

$$T = \sum_{i=1}^{3} \frac{(O_i - E_i)^2}{E_i} \,. \tag{1.7}$$

Under H_0, the random variable T approximately has a χ^2_2 distribution (the number of degrees of freedom is the number of classes, 3, minus 1). This completes the first stage of the empirical test.

In the second stage, $K = 20$ replications of T are generated. The test statistics for the χ^2 test were 2.5556, 11.3314, 146.2747, 24.9729, 1.6850, 50.7449, 2.6507, 12.9015, 40.9470, 8.3449, 11.8191, 9.4470, 91.1219, 37.7246, 18.6256, 1.2965, 1.2267, 0.8346, 23.3909, 14.7596.

Notice that the null hypothesis would not be rejected if it were based only on the first outcome, 2.5556, as the p-value, $\mathbb{P}_{H_0}(T \geqslant 2.5556) \approx 0.279$ is quite large (and therefore the observed outcome is not uncommon under the null hypothesis). However, other values, such as 50.7449 are very large and lead to very small p-values (and a rejection of H_0). The second stage combines these findings into a single number, using a Kolmogorov–Smirnov test, to test whether the distribution of T does indeed follow a χ^2_2 distribution. The empirical cdf (of the 20 values for T) and the cdf of the χ^2_2 distribution are depicted in Figure 1.1. The figure shows a clear disagreement between the two cdfs. The maximal gap between the cdfs is 0.6846 in this case, leading to a Kolmogorov–Smirnov test statistic value of $\sqrt{20} \times 0.6846 \approx 3.06$, which gives a p-value of around 3.7272×10^{-9}, giving overwhelming evidence that the output sequence of the `drand48` generator does not behave like an iid $\mathsf{U}(0,1)$ sequence.

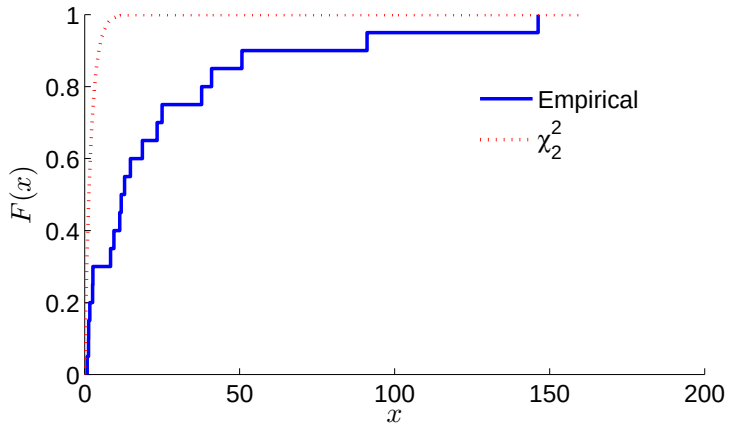


Figure 1.1: Kolmogorov–Smirnov test for the binary rank test using the `drand48` generator.

By comparison, we repeated the same procedure using the default MATLAB generator. The result of the Kolmogorov–Smirnov test is given in Figure 1.2. In this case the empirical and theoretical cdfs have a close match, and the p-value

is large, indicating that the default MATLAB generator passes the binary rank test.

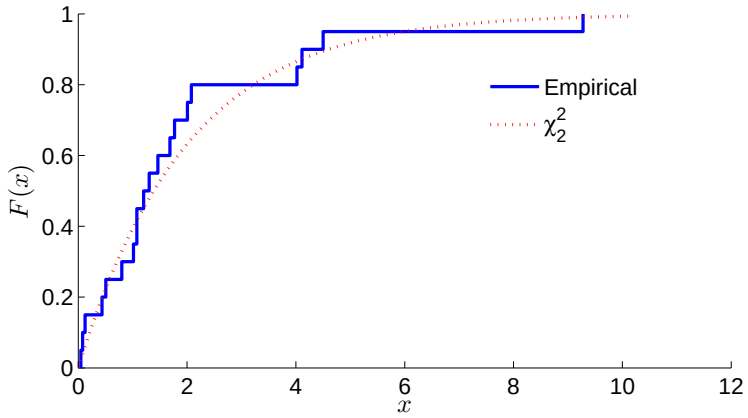


Figure 1.2: Kolmogorov–Smirnov test for the binary rank test using the default MATLAB random number generator (in this case the Mersenne twister).

Today's most complete library for the empirical testing of random number generators is the *TestU01* software library by L'Ecuyer and Simard.

We conclude with a selection of empirical tests. Below, $U_0, U_1, \ldots$ is the original test sequence. The null hypothesis H_0 is that $\{U_i\} \sim_{\text{iid}} \mathsf{U}(0,1)$. Other random variables and processes derived from the $\{U_i\}$ are:

- $Y_0, Y_1, \ldots$, with $Y_i = \lfloor mU_i \rfloor$, $i = 0, 1, \ldots$, for some integer (*size*) $m \geqslant 1$. Under H_0 the $\{Y_i\}$ are iid with a discrete uniform distribution on $\{0, 1, \ldots, m-1\}$.

- $\mathbf{U}_0, \mathbf{U}_1, \ldots$, with $\mathbf{U}_i = (U_{id}, \ldots, U_{id+d-1})$, $i = 0, 1, \ldots$ for some dimension $d \geqslant 1$. Under H_0 the $\{\mathbf{U}_i\}$ are independent random vectors, each uniformly distributed on the d-dimensional hypercube $(0,1)^d$.

- $\mathbf{Y}_0, \mathbf{Y}_1, \ldots$, with $\mathbf{Y}_i = (Y_{id}, \ldots, Y_{id+d-1})$, $i = 0, 1, \ldots$ for some dimension $d \geqslant 1$. Under H_0 the $\{\mathbf{Y}_i\}$ are independent random vectors, each from the discrete uniform distribution on the d-dimensional set $\{0, 1, \ldots, m-1\}^d$.

1.4.1 Equidistribution (or Frequency) Tests

This is to test whether the $\{U_i\}$ have a $\mathsf{U}(0,1)$ distribution. Two possible approaches are:

1. Apply a Kolmogorov–Smirnov test to ascertain whether the empirical cdf of $U_0, \ldots, U_{n-1}$ matches the theoretical cdf of the $\mathsf{U}(0,1)$ distribution; that is, $F(x) = x$, $0 \leqslant x \leqslant 1$.

2. Apply a χ^2 test on $Y_0, \ldots, Y_{n-1}$, comparing for each $k = 0, \ldots, m-1$ the observed number of occurrences in class k, $O_k = \sum_{i=0}^{n-1} \mathrm{I}_{\{Y_i = k\}}$,

with the expected number $E_k = n/m$. Under H_0 the χ^2 statistic (1.7) asymptotically has (as $n \to \infty$) a χ^2_{m-1} distribution.

1.4.2 Serial Tests

This is to test whether successive values of the random number generator are uniformly distributed. More precisely, generate vectors $\mathbf{Y}_0, \ldots, \mathbf{Y}_{n-1}$ for a given dimension d and size m. Count the number of times that the vector $\mathbf{Y}$ satisfies $\mathbf{Y} = \mathbf{y}$, for $\mathbf{y} \in \{0, \ldots, m-1\}^d$, and compare with the expected count n/m^d via a χ^2 goodness of fit test. It is usually recommended that each class should have enough samples, say at least 5 in expectation, so that $n \geqslant 5m^d$. Typically, d is small, say 2 or 3.

1.4.3 Gap Tests

Let $T_1, T_2, \ldots$ denote the times when the output process $U_0, U_1, \ldots$, visits a specified interval $(\alpha, \beta) \subset (0,1)$, and let $Z_1, Z_2, \ldots$ denote the **gap** lengths between subsequent visits; that is, $Z_i = T_i - T_{i-1} - 1$, $i = 1, 2, \ldots$, with $T_0 = 0$. Under H_0, the $\{Z_i\}$ are iid with a $\mathsf{Geom}_0(p)$ distribution, with $p = \beta - \alpha$; that is,

$$\mathbb{P}(Z = z) = p\,(1-p)^z, \quad z = 0, 1, 2, \ldots \,.$$

The gap test assesses this hypothesis by tallying the number of gaps that fall in certain classes. In particular, a χ^2 test is performed with classes $Z = 0, Z = 1, \ldots, Z = r-1$, and $Z \geqslant r$, with probabilities $p(1-p)^z$, $z = 0, \ldots, r-1$ for the first r classes and $(1-p)^r$ for the last class. The integers n and r should be chosen so that the expected number per class is $\geqslant 5$.

When $\alpha = 0$ and $\beta = 1/2$, this is sometimes called **runs above the mean**, and when $\alpha = 1/2$ and $\beta = 1$ this is sometimes called **runs below the mean**.

1.4.4 Poker or Partition Tests

Consider the sequence of d-dimensional vectors $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$, each taking values in $\{0, \ldots, m-1\}^d$. For such a vector $\mathbf{Y}$, let Z be the number of distinct components; for example if $\mathbf{Y} = (4, 2, 6, 4, 2, 5, 1, 4)$, then $Z = 5$. Under H_0, Z has probability distribution

$$\mathbb{P}(Z = z) = \frac{m(m-1)\cdots(m-z+1)\left\{ {d \atop z} \right\}}{m^d}, \quad z = 1, \ldots, \min\{d, m\}\,. \tag{1.8}$$

Here, $\left\{ {d \atop z} \right\}$ represents the **Stirling number of the second kind**, which gives the number of ways a set of size d can be partitioned into z non-empty subsets. For example, $\left\{ {4 \atop 2} \right\} = 7$. Such Stirling numbers can be expressed in terms of binomial coefficients as

$$\left\{ {d \atop z} \right\} = \frac{1}{z!} \sum_{k=0}^{z} (-1)^{z-k} \binom{z}{k} k^d \,.$$

Using the above probabilities, the validity of H_0 can now be tested via a χ^2 test.

1.4.5 Coupon Collector's Tests

Consider the sequence $Y_1, Y_2, \ldots$, each Y_i taking values in $\{0, \ldots, m-1\}$. Let T be the first time that a "complete" set $\{0, \ldots, m-1\}$ is obtained among $Y_1, \ldots, Y_T$. The probability that $(Y_1, \ldots, Y_t)$ is incomplete is, by (1.8), equal to $\mathbb{P}(T > t) = 1 - m! \left\{ {t \atop m} \right\} / m^t$, so that

$$\mathbb{P}(T = t) = \frac{m!}{m^t} \left\{ {t-1 \atop m-1} \right\}, \quad t = m, m+1, \ldots \; .$$

The coupon collector's test proceeds by generating successive times $T_1, \ldots, T_n$ and applying a χ^2 goodness of fit test using classes $T = t$, $t = m, \ldots, r-1$ and $T > r-1$, with probabilities given above.

1.4.6 Permutation Tests

Consider the d-dimensional random vector $\mathbf{U} = (U_1, \ldots, U_d)^\top$. Order the components from smallest to largest and let $\mathbf{\Pi}$ be the corresponding ordering of indices. Under H_0,

$$\mathbb{P}(\mathbf{\Pi} = \boldsymbol{\pi}) = \frac{1}{d!} \quad \text{for all permutations } \boldsymbol{\pi} \; .$$

The permutation test assesses this uniformity of the permutations via a χ^2 goodness of fit test with $d!$ permutation classes, each with class probability $1/d!$.

1.4.7 Run Tests

Consider the sequence $U_1, U_2, \ldots$. Let Z be the **run-up length**; that is, $Z = \min\{k : U_{k+1} < U_k\}$. Under H_0, $\mathbb{P}(Z \geqslant z) = 1/z!$, so that

$$\mathbb{P}(Z = z) = \frac{1}{z!} - \frac{1}{(z+1)!}, \quad z = 1, 2, \ldots \; .$$

In the run test, n of such run lengths $Z_1, \ldots, Z_n$ are obtained, and a χ^2 test is performed on the counts, using the above probabilities. It is important to start from fresh after each run. In practice this is done by throwing away the number immediately after a run. For example the second run is started with U_{Z_1+2} rather than U_{Z_1+1}, since the latter is not $\mathsf{U}(0,1)$ distributed, as it is by definition smaller than U_{Z_1}.

1.4.8 Maximum-of-d Tests

Generate $\mathbf{U}_1, \ldots, \mathbf{U}_n$ for some dimension d. For each $\mathbf{U} = (U_1, \ldots, U_d)^\top$ let $Z = \max\{U_1, \ldots, U_d\}$ be the maximum. Under H_0, Z has cdf

$$F(z) = \mathbb{P}(Z \leqslant z) = z^d, \quad 0 \leqslant z \leqslant 1 \; .$$

Apply the Kolmogorov–Smirnov test to $Z_1, \ldots, Z_n$ with distribution function $F(z)$. Another option is to define $W_k = Z_k^d$ and apply the equidistribution test to $W_1, \ldots, W_n$.

1.4.9 Collision Tests

Consider a sequence of d-dimensional vectors $\mathbf{Y}_1, \ldots, \mathbf{Y}_b$, each taking values in $\{0, \ldots, m-1\}^d$. There are $r = m^d$ possible values for each $\mathbf{Y}$. Typically, $r \gg b$. Think of throwing b balls into r urns. As there are many more urns than balls, most balls will land in an empty urn, but sometimes a "collision" occurs. Let C be the number of such collisions. Under H_0 the probability of c collisions (that is, the probability that exactly $b-c$ urns are occupied) is given, as in (1.8), by

$$\mathbb{P}(C = c) = \frac{r(r-1)\cdots(r-(b-c)+1)\left\{{b \atop b-c}\right\}}{r^b}, \quad c = 0, \ldots, b-1 .$$

A χ^2 goodness of fit test can be applied to compare the empirical distribution of n such collision values, $C_1, \ldots, C_n$, with the above distribution under H_0. One may need to group various of the classes $C = c$ in order to obtain a sufficient number of observations in each class.

1.4.10 Rank of Binary Matrix Tests

Transform the sequence $U_1, U_2, \ldots$ to a binary sequence $B_1, B_2, \ldots$ and arrange these in a binary array of dimension $r \times c$ (assume $r \leqslant c$). Under H_0 the distribution of the rank (in modulo 2 arithmetic) Z of this matrix is given by

$$\mathbb{P}(Z = z) = 2^{(c-z)(z-r)} \prod_{i=0}^{z-1} \frac{(1-2^{i-c})(1-2^{i-r})}{1-2^{i-z}}, \quad z = 0, 1, \ldots, r . \tag{1.9}$$

This can be seen, for example, by defining a Markov chain $\{Z_t, t = 0, 1, 2, \ldots\}$, starting at 0 and with transition probabilities $p_{i,i} = 2^{-c+i}$ and $p_{i,i+1} = 1-2^{-c+i}$, $i = 0, \ldots, r$. The interpretation is that Z_t is the rank of a $t \times c$ matrix which is constructed from a $(t-1) \times c$ matrix by adding a $1 \times c$ random binary row; this row is either dependent on the $t-1$ previous rows (rank stays the same) or not (rank is increased by 1). The distribution of Z_r corresponds to (1.9).

For $c = r = 32$ we have

$$\begin{aligned} \mathbb{P}(Z \leqslant 30) &\approx 0.1336357 \\ \mathbb{P}(Z = 31) &\approx 0.5775762 \\ \mathbb{P}(Z = 32) &\approx 0.2887881 . \end{aligned}$$

These probabilities can be compared with the observed frequencies, via a χ^2 goodness of fit test.

1.4.11 Birthday Spacings Tests

Consider the sequence $Y_1, \ldots, Y_n$ taking values in $\{0, \ldots, m-1\}$. Sort the sequence as $Y_{(1)} \leqslant \ldots \leqslant Y_{(n)}$ and define spacings $S_1 = Y_{(2)} - Y_{(1)}, \ldots, S_{n-1} = Y_{(n)} - Y_{(n-1)}$, and $S_n = Y_{(1)} + m - Y_{(n)}$. Sort the spacings and denote them as $S_{(1)} \leqslant \ldots \leqslant S_{(n)}$.

Let R be the number of times that we have $S_{(j)} = S_{(j-1)}$ for $j = 1, \ldots, n$. The distribution of R depends on m and n, but for example when $m = 2^{25}$ and $n = 512$, we have:

$$\begin{aligned}
\mathbb{P}(R = 0) &\approx 0.368801577 \\
\mathbb{P}(R = 1) &\approx 0.369035243 \\
\mathbb{P}(R = 2) &\approx 0.183471182 \\
\mathbb{P}(R \geqslant 3) &\approx 0.078691997 \,.
\end{aligned}$$

The idea is to repeat the test many times, say $N = 1000$, and perform a χ^2 test on the collected data. Asymptotically, for large n, R has a $\mathsf{Poi}(\lambda)$ distribution, with $\lambda = n^3/(4m)$, where λ should not be large. An alternative is to use $N = 1$ and base the decision whether to reject H_0 or not on the approximate p-value $\mathbb{P}(R \geqslant r) \approx 1 - \sum_{k=0}^{r-1} \mathrm{e}^{-\lambda} \lambda^k / k!$ (reject H_0 for small values). As a rule of thumb the Poisson approximation is accurate when $m \geqslant (4N\lambda)^4$; that is, $Nn^3 \leqslant m^{5/4}$.

1.5 Exercises

1. Implement the C random number generator `drand48` (see Example 1.4). Structure your MATLAB program as follows:

```
function u = drand48(seed)
persistent x    %the state variable x is kept in memory
a =
m =
c =
if (nargin ==0)
  x =
else
  x =
end
u =
```

2. Using the above implementation, generate $N = 100$ "random" numbers $u_1, u_2, \ldots, u_N$ and plot the points $(u_1, u_2), \ldots, (u_{N-1}, u_N)$ in the unit square. Do the points look randomly distributed? Do the same for $N = 1000$.

3. Go to Karl Entacher's "classical LCGs" page: `random.mat.sbg.ac.at/results/karl/server/node4.html`. Choose a random number generator from this page and carry out a two-stage empirical test that shows that the output sequence does not behave like an iid $\mathsf{U}(0, 1)$ sequence.

Chapter 2

Random Variable Generation

Generating a random vector $\mathbf{X}$ from an arbitrary distribution in some Euclidean space $\mathbb{R}^d$ invariably involves the following two steps:

1. Draw uniform random numbers $U_1, \ldots, U_k$, for some $k = 1, 2, \ldots$.
2. Return $\mathbf{X} = g(U_1, \ldots, U_k)$, where g is some function from $(0,1)^k$ to $\mathbb{R}^d$.

The generation of uniform random numbers in the first step is discussed in Chapter 1. The present chapter considers how the second step is implemented. In Section 2.1 we consider various general methods for generating one-dimensional random variables and in Section 2.2 we consider methods for generation of multivariate random variables. Section 2.3 is about generating uniformly in and on a hypersphere. Section 2.4 discusses the uniform generation of permutations. Specific generation algorithms for common discrete and continuous distributions are given in Chapter 3. ☞ 45

All generation methods in this chapter are *exact*, in the sense that each generated random variable has exactly the required distribution (assuming the uniform number generation and computer arithmetic are exact). For an increasing number of Monte Carlo applications exact random variable generation is difficult or impossible to achieve, and *approximate* generation methods are called for, the most prominent being Markov chain Monte Carlo methods; see Chapter 5. ☞ 87

2.1 Generic Algorithms Based on Common Transformations

Many common distributions and families of distributions are related to each other via simple transformations. Such relations lead to general rules for generating random variables. For example, generating random variables from any location–scale family of distributions can be carried out by generating random variables from the base distribution of the family, followed by an affine transformation. A selection of common transformations is discussed in Section 2.1.2. Universal procedures for generating random variables include the inverse-transform method (Section 2.1.1), the alias method (Section 2.1.4), the

composition method (Section 2.1.2.5), and the acceptance–rejection method (Section 2.1.5).

2.1.1 Inverse-Transform Method

Let X be a random variable with cdf F. Since F is a nondecreasing function, the inverse function F^{-1} may be defined as

$$F^{-1}(y) = \inf\{x : F(x) \geqslant y\} \ , \quad 0 \leqslant y \leqslant 1 \ . \tag{2.1}$$

Let $U \sim \mathsf{U}(0,1)$. The cdf of the inverse transform $F^{-1}(U)$ is given by

$$\mathbb{P}(F^{-1}(U) \leqslant x) = \mathbb{P}(U \leqslant F(x)) = F(x) \ . \tag{2.2}$$

Thus, to generate a random variable X with cdf F, draw $U \sim \mathsf{U}(0,1)$ and set $X = F^{-1}(U)$. This leads to the following general method, illustrated in Figure 2.1, for generating from an arbitrary cdf F.

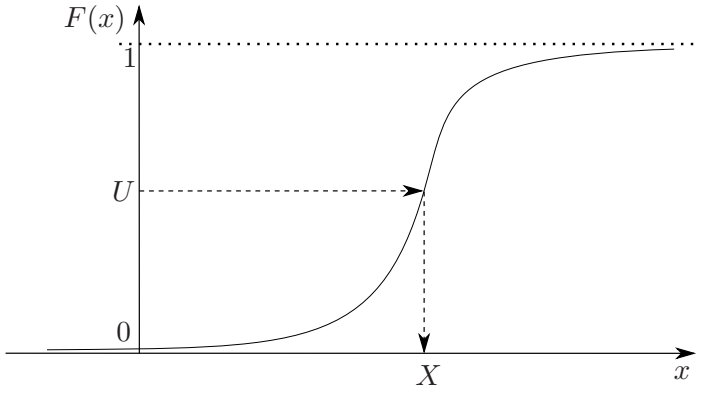


Figure 2.1: Inverse-transform method.

Algorithm 2.1 (Inverse-Transform Method)

1. *Generate* $U \sim \mathsf{U}(0,1)$.
2. *Return* $X = F^{-1}(U)$.

Example 2.1 (Illustration of the Inverse-Transform Method)
Generate a random variable from the pdf

$$f(x) = \begin{cases} 2x, & 0 \leqslant x \leqslant 1 \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The cdf F is defined by $F(x) = \int_0^x 2y \, \mathrm{d}y = x^2$, $0 \leqslant x \leqslant 1$, the inverse function of which is given by $F^{-1}(u) = \sqrt{u}$ for $0 \leqslant u \leqslant 1$. Therefore, to generate a random variable X from the pdf (2.3), first generate a random variable U from $\mathsf{U}(0,1)$, and then take its square root.

In general, the inverse-transform method requires that the underlying cdf, F, exists in a form for which the corresponding inverse function F^{-1} can be found analytically or algorithmically. Applicable distributions are, for example, the exponential, uniform, and Cauchy distributions. Unfortunately, for many other probability distributions, it is either impossible or difficult to find the inverse transform, that is, to solve

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t = u\ ,$$

with respect to x. Even in the case where F^{-1} exists in an explicit form, the inverse-transform method may not necessarily be the most efficient random variable generation method.

The inverse-transform method applies to both absolutely continuous and discrete distributions. For a discrete random variable X taking values $x_1 < x_2 < \ldots$ with probabilities $p_1, p_2, \ldots$, where $\sum_i p_i = 1$, the cdf is a step function, as illustrated in Figure 2.2.

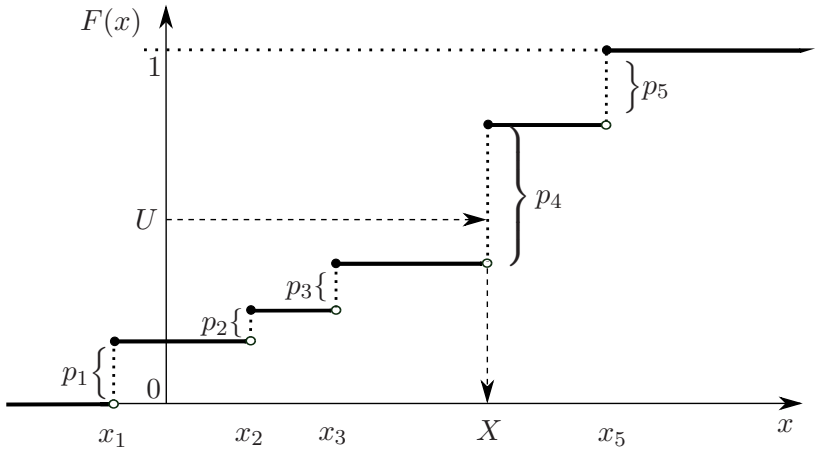


Figure 2.2: Inverse-transform method for a discrete random variable.

For the discrete case the inverse-transform method can be written as follows.

Algorithm 2.2 (Discrete Inverse-Transform Method)

1. *Generate* $U \sim \mathsf{U}(0,1)$.
2. *Find the smallest positive integer* k *such that* $F(x_k) \geqslant U$, *and return* $X = x_k$.

Example 2.2 (Discrete Inverse-Transform Implementation) Suppose we wish to draw $N = 10^5$ independent copies of a discrete random variable taking values $1, \ldots, 5$ with probabilities $0.2, 0.3, 0.1, 0.05, 0.35$, respectively. The following MATLAB program implements the inverse transform method to achieve this, and records the frequencies of occurrences of $1, \ldots, 5$.

```
%discIT.m
p = [0.2,0.3,0.1,0.05,0.35];
N = 10^5;
x = zeros(N,1);
for i=1:N
    x(i) = min(find(rand<cumsum(p))); %draws from p
end
freq = hist(x,1:5)/N
```

Note that `cumsum(p)` corresponds to the vector of cdf values $(F(1), \ldots, F(5))$. By applying the function `find` first and then `min`, one finds the smallest index k such that $F(k) \geqslant$ `rand`, where `rand` presents a uniform random number. A faster generation program, which uses the function `histc(x,e)` to efficiently count the number of values in a vector `x` that fall between the elements of a vector `e`, is given next.

```
%discinvtrans.m
p = [0.2,0.3,0.1,0.05,0.35];
N = 10^5;
[dummy,x]=histc(rand(1,N),[0,cumsum(p)]);
freq = hist(x,1:5)/N
```

2.1.2 Other Transformation Methods

Many distributions used in Monte Carlo simulation are the result of simple operations on random variables. We list some of the main examples.

2.1.2.1 Affine Transformation

Let $\mathbf{X} = (X_1, \ldots, X_n)^\top$ be a random vector, A an $m \times n$ matrix, and $\mathbf{b}$ an $m \times 1$ vector. The $m \times 1$ random vector

$$\mathbf{Z} = A\mathbf{X} + \mathbf{b}$$

is said to be an **affine transformation** of $\mathbf{X}$. If $\mathbf{X}$ has an expectation vector $\boldsymbol{\mu}_{\mathbf{X}}$, then the expectation vector of $\mathbf{Z}$ is $\boldsymbol{\mu}_{\mathbf{Z}} = A\boldsymbol{\mu}_{\mathbf{X}} + \mathbf{b}$. If $\mathbf{X}$ has a covariance matrix $\Sigma_{\mathbf{X}}$, then the covariance matrix of $\mathbf{Z}$ is $\Sigma_{\mathbf{Z}} = A\,\Sigma_{\mathbf{X}}\,A^\top$. Finally, if A is an invertible $n \times n$ matrix and $\mathbf{X}$ has a pdf $f_{\mathbf{X}}$, then the pdf of $\mathbf{Z}$ is given by

$$f_{\mathbf{Z}}(\mathbf{z}) = \frac{f_{\mathbf{X}}(A^{-1}(\mathbf{z} - \mathbf{b}))}{|\det(A)|}, \;\; \mathbf{z} \in \mathbb{R}^n \;,$$

where $|\det(A)|$ denotes the absolute value of the determinant of A.

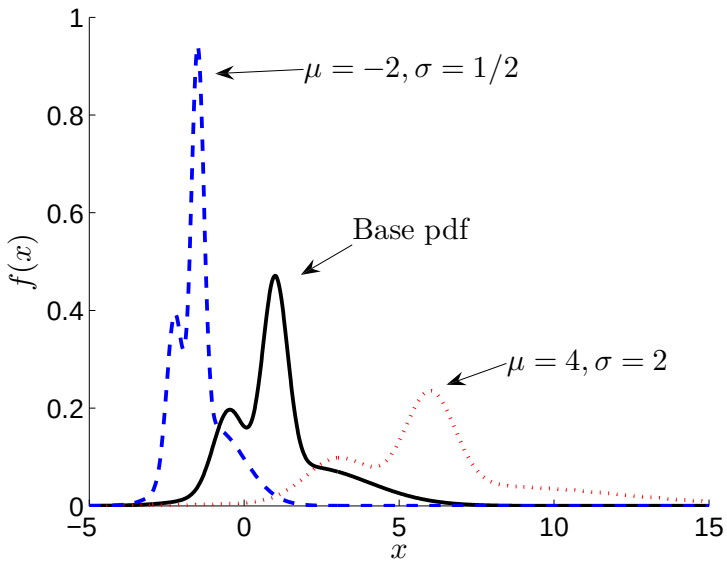


Figure 2.3: A location–scale family of pdfs.

2.1.2.2 Location–Scale Family

A family of continuous distributions with pdfs $\{f(x;\mu,\sigma), \mu \in \mathbb{R}, \sigma > 0\}$ of the form

$$f(x;\mu,\sigma) = \frac{1}{\sigma}\,\mathring{f}\left(\frac{x-\mu}{\sigma}\right), \quad x \in \mathbb{R} \tag{2.4}$$

is called a **location–scale family** with **base** (or standard) pdf $\mathring{f}(x)$. Parameter μ is called the **location** and σ is called the **scale**. Families for which (2.4) holds with $\mu = 0$ are called **scale families**. Families for which (2.4) holds with $\sigma = 1$ are called **location families**.

In a location–scale family the graph of the pdf $f(\cdot;\mu,\sigma)$ has the same shape as that of $\mathring{f}(\cdot)$ but is shifted over a distance μ and scaled by a factor σ, as illustrated in Figure 2.3.

Location–scale families of distributions arise from the affine transformation

$$Z = \mu + \sigma X \ ,$$

where X is distributed according to the base or "standard" pdf of the family. In particular, if $X \sim \mathring{f} \equiv f(\cdot;0,1)$, then

$$\mu + \sigma X \sim f(\cdot;\mu,\sigma) \ .$$

Thus, to generate a random variable from a location–scale family of pdfs, first generate a random variable from the base pdf and then apply an affine transformation to that random variable.

Example 2.3 (Normal Distribution and Location–Scale) A typical example of a location–scale family is the normal family of distributions $\{\mathsf{N}(\mu,\sigma^2)\}$ with location parameter μ and scale parameter σ. Here ☞ 53

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

and $\mathring{f}(x) = (2\pi)^{-1/2}\,\mathrm{e}^{-x^2/2}$ is the base pdf. Hence, to draw $Z \sim \mathsf{N}(\mu,\sigma^2)$, first draw $X \sim \mathsf{N}(0,1)$ and then return $Z = \mu + \sigma X$. In MATLAB, drawing from the standard normal distribution is implemented via the function `randn`. For example, the following MATLAB program draws 10^5 samples from $\mathsf{N}(4,9)$ and plots the corresponding histogram.

```
X = randn(1,10^5);  Z = 4 + 3*X;   hist(Z,100)
```

2.1.2.3 Reciprocation

Another common transformation is inversion or reciprocation. Specifically, if X is a univariate random variable, then the **inverse** or **reciprocal** of X is

$$Z = \frac{1}{X}\,.$$

If X has pdf f_X, then Z has pdf

$$f_Z(z) = \frac{f_X(z^{-1})}{z^2},\ \ z \in \mathbb{R}\,. \tag{2.5}$$

Distributions obtained in this way are called **inverted** or **inverse distributions**.

Example 2.4 (Inverse-Gamma Distribution via Reciprocation) The **inverse-gamma** distribution, denoted by $\mathsf{InvGamma}(\alpha,\lambda)$, has pdf

$$f_Z(z;\alpha,\lambda) = \frac{\lambda^\alpha z^{-\alpha-1}\mathrm{e}^{-\lambda z^{-1}}}{\Gamma(\alpha)},\ \ z>0\,,$$

☞ 51 which is of the form (2.5), with f_X the pdf of the $\mathsf{Gamma}(\alpha,\lambda)$ distribution. To generate a random variable $Z \sim \mathsf{InvGamma}(\alpha,\lambda)$, draw $X \sim \mathsf{Gamma}(\alpha,\lambda)$ and return $Z = 1/X$.

2.1.2.4 Truncation

Let $\mathsf{Dist}_{\mathscr{A}}$ and $\mathsf{Dist}_{\mathscr{B}}$ be two distributions on sets $\mathscr{A}$ and $\mathscr{B} \subset \mathscr{A}$, respectively. Let $\mathbf{X} \sim \mathsf{Dist}_{\mathscr{A}}$ and $\mathbf{Z} \sim \mathsf{Dist}_{\mathscr{B}}$. If the conditional distribution of $\mathbf{X}$ given $\mathbf{X} \in \mathscr{B}$ coincides with the distribution of $\mathbf{Z}$ (that is, $\mathsf{Dist}_{\mathscr{B}}$), then the latter distribution is said to be the **truncation** of $\mathsf{Dist}_{\mathscr{A}}$ to $\mathscr{B}$. In particular, if $f_{\mathbf{X}}$ is the pdf of $\mathbf{X}$, then the pdf of $\mathbf{Z}$ is (in the continuous case)

$$f_{\mathbf{Z}}(\mathbf{z}) = \frac{f_{\mathbf{X}}(\mathbf{z})}{\int_{\mathscr{B}} f_{\mathbf{X}}(\mathbf{x})\,\mathrm{d}\mathbf{x}},\quad \mathbf{z} \in \mathscr{B}\,.$$

In the continuous univariate case, the truncation of a pdf $f(x)$ to an interval $[a, b]$ gives the pdf

$$f_Z(z) = \frac{f(z)}{\int_a^b f(x)\,\mathrm{d}x}, \quad a \leqslant z \leqslant b\,,$$

and in the discrete case we replace the integral with a sum. In terms of cdfs we have:

$$F_Z(z) = \frac{F(z) - F(a-)}{F(b) - F(a-)}, \quad a \leqslant z \leqslant b\,, \tag{2.6}$$

where $F(a-) = \lim_{x \uparrow a} F(x)$. To generate random variables from a truncated distribution on $[a, b]$ one can simply use the acceptance–rejection method (see Section 2.1.5) by generating $X \sim F$ until $X \in [a, b]$. When the generation of X can be readily performed via the inverse-transform method, a more direct approach can be taken. In particular, the inverse of (2.6) yields the following inverse-transform method.

Algorithm 2.3 (Truncation via the Inverse-Transform Method)

1. *Generate* $U \sim \mathsf{U}(0, 1)$.
2. *Return* $Z = F^{-1}(F(a-) + U(F(b) - F(a-)))$.

Note that the only difference with the inverse-transform method is that in Step 2 the argument of F^{-1} is uniformly distributed on the interval $(F(a-), F(b))$ rather than on $(0, 1)$.

Example 2.5 (Truncated Exponential Generator) Consider the pdf of the $\mathsf{Exp}(1)$ distribution truncated to the interval $[0, 2]$: ☞ 51

$$f_Z(z) = \frac{\mathrm{e}^{-z}}{1 - \mathrm{e}^{-2}}, \quad 0 \leqslant z \leqslant 2\,. \tag{2.7}$$

The inverse of the cdf of the $\mathsf{Exp}(1)$ distribution is $F^{-1}(u) = -\ln(1 - u)$, so that

$$Z = -\ln(1 + U(\mathrm{e}^{-2} - 1)) \sim f_Z\,.$$

The following MATLAB program provides an implementation for generating 10^5 samples from this truncated distribution and plotting the corresponding histogram.

```
%truncexp.m
U= rand(1,10^5); Z = -log(1 + U *(exp(-2) - 1)); hist(Z,100)
```

Example 2.6 (Truncated Normal Generator) Consider the $\mathsf{N}(\mu, \sigma^2)$ pdf truncated to the interval $[a, b]$:

$$f_Z(z) = \frac{1}{\sigma\, C}\, \varphi\left(\frac{z - \mu}{\sigma}\right), \quad a \leqslant z \leqslant b\,,$$

where $C = \Phi\big(\frac{b-\mu}{\sigma}\big) - \Phi\big(\frac{a-\mu}{\sigma}\big)$, and φ and Φ are the pdf and cdf of the $\mathsf{N}(0,1)$ distribution, respectively. The following MATLAB function implements the inverse-transform method.

```
function out=normt(mu,sig,a,b)
pb=normcdf((b-mu)./sig);
pa=normcdf((a-mu)./sig);
C=pb-pa;
out=mu+sig.*norminv(C.*rand(size(mu))+pa);
```

Example 2.7 (Sampling from the Tail of a Normal Distribution) Consider the problem of sampling from the truncated normal pdf

$$f_Z(z) = \frac{\varphi(z)\,\mathrm{I}_{\{z \geqslant a\}}}{\Phi(-a)} \,,$$

where the truncation point $a > 0$ is large, say $a > 10$. A straightforward implementation of the inverse-transform method gives:

$$Z = \Phi^{-1}(\Phi(a) + U\,(1 - \Phi(a))), \quad U \sim \mathsf{U}[0,1] \,.$$

However, this approach is numerically unstable, and in most computer implementations one obtains infinity for the value of Z or an error message when $a > 6.4$. A theoretically equivalent but more numerically stable generator is:

$$Z = -\Phi^{-1}(U\,\Phi(-a)), \quad U \sim \mathsf{U}[0,1] \,.$$

This generator works well for values of a up to $a = 37$. However, it still breaks down in MATLAB for values of $a > 37$. The improved reliability is due to the fact that it is easier to approximate Φ^{-1} in the left tail than in the right tail. This example shows that Algorithm 2.3 should be used with caution and is not prescriptive for all problems.

2.1.2.5 Composition Method

Of great practical importance are distributions that are probabilistic mixtures of other distributions. Let $\mathscr{T}$ be an index set and $\{H_t, t \in \mathscr{T}\}$ be a collection of cdfs (possibly multidimensional). Suppose that G is the cdf of a distribution on $\mathscr{T}$. Then

$$F(\mathbf{x}) = \int_{\mathscr{T}} H_t(\mathbf{x})\,\mathrm{d}G(t) \,,$$

is again a cdf and the corresponding distribution is called a **mixture distribution** or simply **mixture**, with **mixing components** $\{H_t, t \in \mathscr{T}\}$. It is useful to think of G as the cdf of a random variable T and H_t as the conditional cdf of a random variable $\mathbf{X}_t$ given $T = t$. Then, F is cdf of the random variable $\mathbf{X}_T$. In other words, if $T \sim G$ and $X_t \sim H_t$, then $X = X_T$ has cdf F. This yields the following generator.

Algorithm 2.4 (Composition Method Generator)

1. *Generate the random variable T according to the cdf G.*
2. *Given $T = t$, generate X from the cdf H_t.*

In many applications G is a distribution on $\{1, \ldots, n\}$ for some strictly positive integer n, in which case the mixture cdf is of the form $F(x) = \sum_{t=1}^{n} p_t F_t(x)$ for some collection of cdfs $\{F_t\}$ and probabilities $\{p_t\}$ summing to 1. Denoting the corresponding pdfs by $\{f_t\}$, the pdf f of the finite mixture is given by

$$f(x) = \sum_{t=1}^{n} p_t f_t(x) \,. \tag{2.8}$$

Example 2.8 (Mixture of Normals) We wish to draw samples from a mixture of normal pdfs. Specifically, suppose that the pdf from which to draw has the form (2.8) with $n = 3$ and $(p_1, p_2, p_3) = (0.2, 0.4, 0.4)$, and suppose that the means and standard deviations of the normal pdfs are given by $\boldsymbol{\mu} = (-0.5, 1, 2)$ and $\boldsymbol{\sigma} = (0.5, 0.4, 2)$. A useful shorthand notation for this distribution is

$$0.2\,\mathsf{N}(-0.5, 0.5^2) + 0.4\,\mathsf{N}(1, 0.4^2) + 0.4\,\mathsf{N}(2, 2^2) \,. \tag{2.9}$$

A graph of the corresponding pdf is given as the base pdf in Figure 2.3. The following MATLAB code implements the composition method and plots a histogram of the generated data.

```
%mixturefin.m
p = [0.2, 0.4, 0.4];
mu = [-0.5, 1, 2];
sigma = [0.5, 0.4, 2];
N = 10^5;
[dummy,t]=histc(rand(1,N),[0,cumsum(p)]); % draw from p
x = randn(1,N).*sigma(t) + mu(t);     % draw a normal r.v.
hist(x,200)                % make a histogram of the data
```

Example 2.9 (Composition Method in Bayesian Inference) Composition methods appear often in Bayesian analysis. As an example, consider the following Bayesian model for a coin toss experiment. Let θ (random) denote the probability of success (heads) and let X be the number of successes in n tosses. Define the joint distribution of X and θ via the hierarchical model

$$\begin{aligned} \theta &\sim \mathsf{Beta}(\alpha, \beta) && \text{prior distribution,} \\ (X \,|\, \theta) &\sim \mathsf{Bin}(n, \theta) && \text{likelihood distribution} \end{aligned}$$

for some given $\alpha > 0$ and $\beta > 0$. Using Bayesian notation, we can write for the pdf of X:

$$f(x) = \int f(x \,|\, \theta)\, f(\theta)\, \mathrm{d}\theta, \quad x = 0, \ldots, n \,,$$

where $f(\theta)$ is the pdf of the $\mathsf{Beta}(\alpha, \beta)$ distribution and $f(x \,|\, \theta)$ is the pdf of the $\mathsf{Bin}(n, \theta)$ distribution. Note that the distribution of X is a continuous mixture. The mechanism for simulating samples from this distribution using the composition method is given precisely in the Bayesian hierarchical model: first draw θ from $\mathsf{Beta}(\alpha, \beta)$, and then, given θ, draw X from $\mathsf{Bin}(n, \theta)$.

2.1.2.6 Polar Transformation

The **polar method** is based on the polar coordinate transformation $X = R\cos\Theta$, $Y = R\sin\Theta$, where $\Theta \sim \mathsf{U}(0, 2\pi)$ and $R \sim f_R$ are independent. Using standard transformation rules it follows that the joint pdf of X and Y satisfies

$$f_{X,Y}(x, y) = \frac{f_R(r)}{2\pi r} \,,$$

with $r = \sqrt{x^2 + y^2}$, so that

$$f_X(x) = \int_0^\infty \frac{f_R(\sqrt{x^2 + y^2})}{\pi\sqrt{x^2 + y^2}} \,\mathrm{d}y \,.$$

For example, if $f_R(r) = r\,\mathrm{e}^{-r^2/2}$, then $f_X(x) = \mathrm{e}^{-x^2/2}/\sqrt{2\pi}$. Note that in this case the pdf of R is the same as that of $\sqrt{2E}$ with $E \sim \mathsf{Exp}(1)$. Equivalently, R has the same distribution as $\sqrt{-2\ln U}$ with $U \sim \mathsf{U}(0, 1)$. These observations lead to the *Box–Muller* method for generating standard normal random variables.

☞ 54

Interesting relationships between distributions can be obtained from a slight modification of the polar method. Specifically, suppose $R \in [0, \infty)$ and $Z_1, Z_2 \sim_{\text{iid}} \mathsf{N}(0, 1)$ are independent random variables. Then, $(X_1, X_2) = R\,(Z_1, Z_2) = (RZ_1, RZ_2)$ has a radially symmetric pdf with radius distributed according to the distribution of $R\sqrt{Z_1^2 + Z_2^2}$, or, equivalently, according to the distribution of $R\sqrt{2E}$, where $E \sim \mathsf{Exp}(1)$ is independent of R. For some choices of R the pdf of $R\sqrt{2E}$ is easy, leading to simple generation algorithms for X_1.

2.1.3 Table Lookup Method

One of the easiest and fastest general methods for generating discrete random variables is Marsaglia's **table lookup method**.

Algorithm 2.5 (Table Lookup Method)

1. *Draw* $U \sim \mathsf{U}(0, 1)$.
2. *Set* $I = \lceil Un \rceil$.
3. *Return* $\mathbf{X} = a_I$.

Here $(a_1, \ldots, a_n)$ is a predetermined table of numbers or, more generally, objects such as vectors, trees, etc. Duplication among the $\{a_i\}$ is allowed. If the set of

distinct objects is $\{b_1, \ldots, b_k\}$, then the algorithm generates random variables $\mathbf{X}$ that satisfy

$$\mathbb{P}(\mathbf{X} = b_i) = \frac{\sum_{j=1}^{n} \mathrm{I}_{\{a_j = b_i\}}}{n} = \frac{\#\{j : a_j = b_i\}}{n}, \quad i = 1, \ldots, k \,.$$

Example 2.10 (Random Variable Generation via Table Lookup) Suppose we wish to generate from the discrete pdf f with

$$f(x) = \frac{x}{55}, \quad x = 1, \ldots, 10 \,.$$

This can be done via table lookup using a table of size $n = 55$ with elements $1, 2, 2, 3, 3, 3, \ldots, 10, \ldots, 10$. The following MATLAB program creates the lookup table, generates 10^5 random variables from f via the lookup method, and plots the histogram of the generated data.

```
%tablook.m
r = 10;
a = zeros(1,(r+1)*r/2);
n=0;
for i=1:r
    for j=1:i
        n = n+1;
        a(n) = i;
    end
end
I = ceil(rand(1,10^5)*n);
X = a(I);
hist(X,1:r)
```

The table lookup method is a *resampling* technique: given data $\{a_i\}$ the algorithm resamples the data by selecting one of the a_i uniformly and independently each time. In other words, Algorithm 2.5 generates samples from the empirical distribution of the data $\{a_i\}$.

2.1.4 Alias Method

The **alias method** is an alternative to the inverse-transform method for generating discrete random variables, which does not require time-consuming search techniques as per Step 2 of Algorithm 2.2. It is based on the fact that an arbitrary n-point distribution can be represented as an equally weighted mixture of n two-point distributions. The idea is to redistribute the probability mass into n bins of equal weight $1/n$, as illustrated in Figure 2.4.

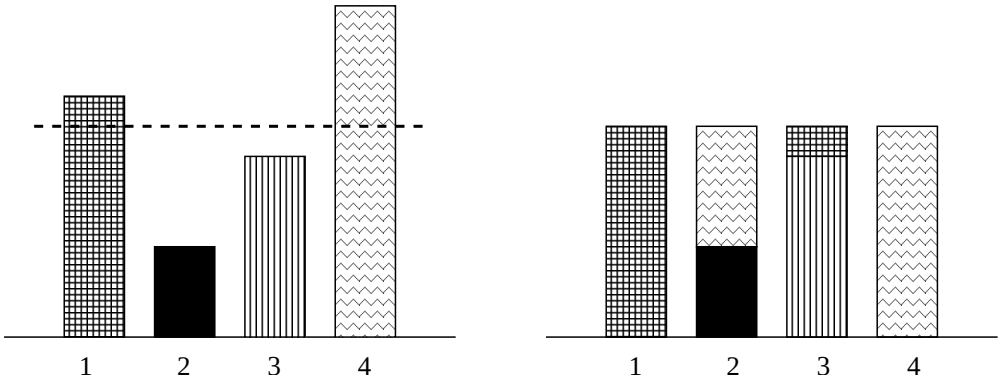


Figure 2.4: Redistribution of probability mass.

Here, a probability distribution on $\{1, 2, 3, 4\}$ is depicted on the left side, with probability masses $8/28, 3/28, 6/28$, and $11/28$. These masses are redistributed over four bins such that (1) the total capacity of each bin is $1/4$, (2) each bin has masses corresponding to at most two variables, (3) bin i contains mass corresponding to variable i, $i = 1, 2, 3, 4$.

To see that such a redistribution can be done generally, consider a probability distribution on $\{1, \ldots, n\}$ with probability mass $p_i > 0$ assigned to i, $i = 1, \ldots, n$. If $p_1 = \ldots = p_n$, then, trivially, the original distribution is an equal mixture of 1-point (and hence 2-point) distributions. If not all $\{p_k\}$ are equal, then there must exist indices i and j such that $p_i < 1/n$ and $p_j \geqslant 1/n$. Now fill bin i by first adding p_i and then transferring an amount $1/n - p_i$ from p_j. This leaves $n - 1$ bins to be filled with $n - 1$ probabilities that sum up to $(n - 1)/n$, which can be done in exactly the same way by choosing i' and j' from the remaining indices such that $p_{i'} < 1/n$ and $p_{j'} \geqslant 1/n$, and redistributing their weights, and so on. At the end, each bin $k = 1, \ldots, n$ corresponds to a 2-point distribution at the points k and another point a_k, with probabilities q_k and $1 - q_k$, respectively. For example, in Figure 2.4, $a_2 = 4$ and $q_2 = 3/28 \times 4 = 3/7$. The $\{a_k\}$ are called the **alias** values and the $\{q_k\}$ the **cut-off** values. These can be determined by the following algorithm, which formalizes the bin-filling procedure described above.

Algorithm 2.6 (Set-up for the Alias Method) *Let $\{p_k, k = 1, \ldots, n\}$ be a distribution on $\{1, \ldots, n\}$.*

1. *Let $q_k = n\, p_k, k = 1, \ldots, n$. Let $\mathscr{S} = \{k : q_k < 1\}$ and $\mathscr{G} = \{k : q_k \geqslant 1\}$.*
2. *While $\mathscr{S}$ and $\mathscr{G}$ are not empty,*
   (a) *Choose some $i \in \mathscr{S}$ and $j \in \mathscr{G}$.*
   (b) *Set $a_i = j$ and $q_j = q_j - (1 - q_i)$.*
   (c) *If $q_j < 1$, remove j from $\mathscr{G}$ and add to $\mathscr{S}$.*
   (d) *Remove i from $\mathscr{S}$.*

The set-up algorithm can be implemented to run in $\mathcal{O}(n)$ time. Once the alias and cut-off values have been established, generation of a random variable X from the distribution $\{p_k\}$ is simple and can be written as follows.

Algorithm 2.7 (Alias Method)

1. *Generate* $U \sim \mathsf{U}(0,1)$ *and set* $K = \lceil n\,U \rceil$.
2. *Draw* $V \sim \mathsf{U}(0,1)$. *If* $V \leqslant q_K$, *return* $X = K$; *otherwise, return* $X = a_K$.

Example 2.11 (Alias Method) The following MATLAB program shows how the alias method works in practice. The objective is to generate 10^6 samples from a fixed 400-point pdf that is itself randomly generated. In the first part of the program the alias and cut-off values are calculated. The second part checks that the original probabilities are faithfully reconstructed. In the last part the data are generated.

```
%aliasfin.m
p =rand(1,400);p = p/sum(p); %the sampling distribution
n = size(p,2);
a = 1:n; %alias values
q = zeros(1,n); % cut-off values
q = n*p;
greater = find(q >= 1);
smaller = find(q < 1);
while (~isempty(smaller) && ~isempty(greater))
    i = smaller(1);
    j = greater(1);
    a(i) = j;
    q(j) = q(j) -(1- q(i));
    if (q(j) < 1)
        greater = setdiff(greater,j);
        smaller = union(smaller,j);
    end
    smaller = setdiff(smaller,i);
end
pp = q/n;
for i = 1:n
    ind = find(a == i);
    pp(i) = pp(i) + sum((1 - q(ind)))/n;
end
max(abs(pp - p))
N = 10^6; % generate sample of  size N
X = zeros(1,N);
for i = 1:N
    K = ceil(rand*n);
    if (rand > q(K));
        X(i) = a(K);
    else
        X(i) = K;
    end
end
```

2.1.5 Acceptance–Rejection Method

The acceptance–rejection method is one of the most useful general methods for sampling from general distributions. It can be applied to both discrete and continuous distributions, and even to multidimensional distributions — although its efficiency rapidly decreases in the number of dimensions (see Section 2.3). The method is based on the following observation.

Theorem 2.1.1 (Acceptance–Rejection) *Let $f(\mathbf{x})$ and $g(\mathbf{x})$ be two pdfs such that for some $C \geqslant 1$, $C\,g(\mathbf{x}) \geqslant f(\mathbf{x})$ for all $\mathbf{x}$. Let $\mathbf{X} \sim g(\mathbf{x})$ and $U \sim \mathsf{U}(0,1)$ be independent. Then, the conditional pdf of $\mathbf{X}$ given $U \leqslant f(\mathbf{X})/(Cg(\mathbf{X}))$ is $f(\mathbf{x})$.*

Proof: Consider the joint pdf of $\mathbf{X}$ and U, which is

$$f_{\mathbf{X},U}(\mathbf{x},u) = \frac{g(\mathbf{x})\,\mathrm{I}_{\left\{u \leqslant \frac{f(\mathbf{x})}{Cg(\mathbf{x})}\right\}}}{\int\int_0^1 g(\mathbf{x})\,\mathrm{I}_{\left\{u \leqslant \frac{f(\mathbf{x})}{Cg(\mathbf{x})}\right\}}\,\mathrm{d}u\,\mathrm{d}\mathbf{x}} = \frac{g(\mathbf{x})\,\mathrm{I}_{\left\{u \leqslant \frac{f(\mathbf{x})}{Cg(\mathbf{x})}\right\}}}{\int g(\mathbf{x})\Big(\int_0^{\frac{f(\mathbf{x})}{Cg(\mathbf{x})}} 1\,\mathrm{d}u\Big)\mathrm{d}\mathbf{x}}$$
$$= C\,g(\mathbf{x})\,\mathrm{I}_{\left\{u \leqslant \frac{f(\mathbf{x})}{Cg(\mathbf{x})}\right\}}\,.$$

The (marginal) pdf of $\mathbf{X}$ is therefore

$$f_{\mathbf{X}}(\mathbf{x}) = \int_0^1 f_{\mathbf{X},U}(\mathbf{x},u)\,\mathrm{d}u = C\,g(\mathbf{x})\frac{f(\mathbf{x})}{Cg(\mathbf{x})} = f(\mathbf{x})\,,$$

as required.

We call $g(\mathbf{x})$ the **proposal** pdf and assume that it is easy to generate random variables from it. The acceptance–rejection method can be formulated as follows.

Algorithm 2.8 (Acceptance–Rejection)

1. *Generate $\mathbf{X}$ from $g(\mathbf{x})$.*
2. *Generate U from $\mathsf{U}(0,1)$, independently of $\mathbf{X}$.*
3. *If $U \leqslant f(\mathbf{X})/(C\,g(\mathbf{X}))$ output $\mathbf{X}$; otherwise, return to Step 1.*

In other words, generate $\mathbf{X} \sim g$ and accept it with probability $f(\mathbf{X})/(C\,g(\mathbf{X}))$; otherwise, reject $\mathbf{X}$ and try again.

The **efficiency** of the acceptance–rejection method is defined as the probability of acceptance, which is,

$$\mathbb{P}\left(U \leqslant \frac{f(\mathbf{X})}{Cg(\mathbf{X})}\right) = \int g(\mathbf{x}) \int_0^1 \mathrm{I}_{\left\{u \leqslant \frac{f(\mathbf{x})}{Cg(\mathbf{x})}\right\}}\mathrm{d}u\,\mathrm{d}\mathbf{x} = \int \frac{f(\mathbf{x})}{C}\,\mathrm{d}\mathbf{x} = \frac{1}{C}\,.$$

Since the trials are independent, the number of trials required to obtain a successful pair $(\mathbf{X}, U)$ has a $\mathsf{Geom}(1/C)$ distribution, so that the expected number of trials is equal to C.

Example 2.12 (Generating from the Positive Normal Distribution) Suppose we wish to generate random variables from the **positive normal** pdf

$$f(x) = \sqrt{\frac{2}{\pi}}\, \mathrm{e}^{-x^2/2}, \qquad x \geqslant 0\,, \tag{2.10}$$

using acceptance–rejection. We can bound $f(x)$ by $C\,g(x)$, where $g(x) = \mathrm{e}^{-x}$ is the pdf of the $\mathsf{Exp}(1)$ distribution. The smallest constant C such that $f(x) \leqslant Cg(x)$ is $\sqrt{2\mathrm{e}/\pi}$. The pdf $f(x)$ and the dominating function $Cg(x)$ are depicted in Figure 2.5. The efficiency of this method is $\sqrt{\pi/2\mathrm{e}} \approx 0.76$.

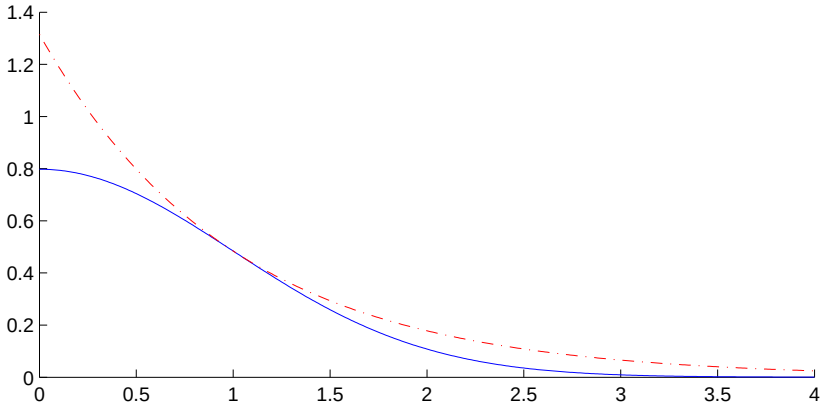


Figure 2.5: Bounding the positive normal density (solid curve).

Since $f(x)$ is the pdf of the absolute value of a standard normal random variable, we can generate $Z \sim \mathsf{N}(0,1)$ by first generating $X \sim f$ as above and then returning $Z = XS$, where S is a random sign; for example, $S = 1 - 2\mathrm{I}_{\{U \leqslant 1/2\}}$ with $U \sim \mathsf{U}(0,1)$. This procedure for generating $\mathsf{N}(0,1)$ random variables is summarized in Algorithm 3.13. ☞ 54

2.2 Generation Methods for Multivariate Random Variables

In this section we consider some general procedures for generating a random vector $\mathbf{X} = (X_1, \ldots, X_n)^\top$ from a given n-dimensional distribution with pdf $f(\mathbf{x})$. Algorithms for generating from specific multivariate distributions are given in Section 3.3. ☞ 55

When the components $X_1, \ldots, X_n$ are *independent* the situation is easy. Suppose that the component pdfs are f_i, $i = 1, \ldots, n$, so that $f(\mathbf{x}) = f_1(x_1) \cdots f_n(x_n)$. To generate $\mathbf{X}$, simply generate each component $X_i \sim f_i$ individually — for example, via the inverse-transform method or acceptance–rejection.

Algorithm 2.9 (Independent Components Generation)

1. *Independently generate* $X_i \sim f_i$, $i = 1, \ldots, n$.
2. *Return* $\mathbf{X} = (X_1, \ldots, X_n)^\top$.

For *dependent* components $X_1, \ldots, X_n$, we can represent the joint pdf $f(\mathbf{x})$ as

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = f_1(x_1)\, f_2(x_2 \,|\, x_1) \cdots f_n(x_n \,|\, x_1, \ldots, x_{n-1}) \;, \qquad (2.11)$$

where $f_1(x_1)$ is the marginal pdf of X_1 and $f_k(x_k \,|\, x_1, \ldots, x_{k-1})$ is the conditional pdf of X_k given $X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}$. This observation leads to the following procedure.

Algorithm 2.10 (Dependent Components Generation)

1. *Generate* $X_1 \sim f_1$. *Set* $t = 1$.
2. *While* $t < n$, *given* $X_1 = x_1, \ldots, X_t = x_t$, *generate* $X_{t+1} \sim f_{t+1}(x_{t+1} \,|\, x_1, \ldots, x_t)$ *and set* $t = t + 1$.
3. *Return* $\mathbf{X} = (X_1, \ldots, X_n)^\top$.

The applicability of this approach depends, of course, on knowledge of the conditional distributions. In certain models, for example Markov models, this
☞ 63
knowledge is easily obtainable.

Another, usually simpler, approach is to generate the random vector $\mathbf{X}$ by multidimensional acceptance–rejection; for instance, when generating a random vector uniformly over an n-dimensional region.

For high-dimensional distributions, efficient *exact* random variable generation is often difficult to achieve, and *approximate* generation methods are used instead. Such methods are discussed in Chapter 5.

2.3 Generating Random Vectors Uniformly Distributed in a Unit Hyperball and Hypersphere

Consider the n-dimensional unit hyperball, $\mathscr{B}_n = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leqslant 1\}$. Generating uniform random vectors in $\mathscr{B}_n$ is straightforward via the acceptance–rejection method.

Algorithm 2.11 (Generating Uniformly in $\mathscr{B}_n$ (I))

1. *Generate* $U_1, \ldots, U_n \overset{\text{iid}}{\sim} \mathsf{U}(0, 1)$.
2. *Set* $X_1 = 1 - 2U_1, \ldots, X_n = 1 - 2U_n$, *and* $R = \sum_{i=1}^n X_i^2$.
3. *If* $R \leqslant 1$, *accept* $\mathbf{X} = (X_1, \ldots, X_n)^\top$ *as the desired vector; otherwise, go to Step 1.*

The efficiency of this n-dimensional acceptance–rejection method is equal to the ratio

$$\frac{1}{C} = \frac{\text{volume of the hyperball}}{\text{volume of the hypercube}} = \frac{\frac{\pi^{n/2}}{(n/2)\Gamma(n/2)}}{2^n} = \frac{1}{n\,2^{n-1}}\frac{\pi^{n/2}}{\Gamma(n/2)} ,$$

which rapidly decreases to 0 as $n \to \infty$; for example, for $n = 8$ the efficiency is approximately 0.016. The next algorithm is more efficient for higher dimensions, and utilizes the following facts.

- If $X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$, then the normalized vector

$$\mathbf{Y} = \left(\frac{X_1}{\|\mathbf{X}\|} , \ldots , \frac{X_n}{\|\mathbf{X}\|} \right), \tag{2.12}$$

  where $\|\mathbf{X}\| = (\sum_{i=1}^n X_i^2)^{\frac{1}{2}}$, is distributed uniformly on the n-dimensional hypersphere $\mathscr{S}_n = \{\mathbf{y} : \|\mathbf{y}\| = 1\}$.

- The radius R of a uniformly chosen point in $\mathscr{B}_n$ has cdf $F_R(r) = r^n$, $0 \leqslant r \leqslant 1$.

Algorithm 2.12 (Generating Uniformly in $\mathscr{B}_n$ (II))

1. *Generate a random vector* $\mathbf{X} = (X_1, \ldots, X_n)^\top$ *with iid* $\mathsf{N}(0,1)$ *components.*
2. *Generate* $U \sim \mathsf{U}(0,1)$ *and set* $R = U^{1/n}$.
3. *Return* $\mathbf{Z} = R\,\mathbf{X}/\|\mathbf{X}\|$.

To generate a random vector that is uniformly distributed over the *surface* of an n-dimensional unit ball — in other words, uniformly on the unit hypersphere $\mathscr{S}_n$, we simplify the previous algorithm and arrive at the following one.

Algorithm 2.13 (Generating Uniform Random Vectors on $\mathscr{S}_n$)

1. *Generate a random vector* $\mathbf{X} = (X_1, \ldots, X_n)^\top$ *with iid* $\mathsf{N}(0,1)$ *components.*
2. *Return* $\mathbf{Y} = \mathbf{X}/\|\mathbf{X}\|$.

2.4 Generating Random Permutations

Suppose we have a collection of n objects, labeled $1, 2, \ldots, n$, and we wish to generate a random permutation of these labels such that each of the $n!$ possible orderings occurs with equal probability. A simple algorithm for generating such uniform permutations is based on the ordering of uniform random numbers.

Algorithm 2.14 (Generating Random Permutations by Sorting)

1. *Generate* $U_1, \ldots, U_n \overset{\text{iid}}{\sim} \mathsf{U}(0,1)$.
2. *Sort these in increasing order:* $U_{X_1} \leqslant U_{X_2} \leqslant \cdots \leqslant U_{X_n}$.
3. *Return* $\mathbf{X} = (X_1, \ldots, X_n)$.

Example 2.13 (Drawing Without Replacement) Suppose we wish to select 30 numbers out of 100 uniformly without replacement. This can be accomplished by generating a uniform random permutation of $1, \ldots, 100$ and selecting the first 30 components thereof. The following MATLAB program achieves this via an implementation of Algorithm 2.14. This procedure is most efficient when the number of draws k is close to n.

```
%unifperm.m
n = 100;
k = 30;
[s,ix] = sort(rand(1,n));
x = ix(1:k)
```

The next algorithm for drawing uniform random permutation is faster than Algorithm 2.14 and builds the permutation component by component, requiring only n uniform random numbers and no sorting.

Algorithm 2.15 (Generating Uniform Random Permutations)

1. *Set* $\mathbf{a} = (1, \ldots, n)$ *and* $i = 1$.
2. *Generate an index* I *uniformly from* $\{1, \ldots, n - i + 1\}$.
3. *Set* $X_i = a_I$ *followed by setting* $a_I = a_{n-i+1}$.
4. *Set* $i = i + 1$. *If* $i \leqslant n$ *go to Step 2.*
5. *Return* $\mathbf{X} = (X_1, \ldots, X_n)$.

2.5 Exercises

1. Let the random variable X have pdf

$$f(x) = \begin{cases} \frac{1}{2}\, x\,, & 0 < x < 1 \\ \frac{1}{2}\,, & 1 \leqslant x \leqslant \frac{5}{2}\,. \end{cases}$$

Generate a random variable from $f(x)$, using the inverse-transform method and the acceptance-rejection method. For the latter use the proposal density

$$g(x) = \frac{8}{25}\, x\,, \quad 0 \leqslant x \leqslant \frac{5}{2}\,.$$

2. Implement an acceptance-rejection and an inverse-transform algorithm for generating random variables from the pdf $f(x) = \sin(x)/2, 0 \leqslant x \leqslant \pi$.

3. We wish to sample from a pdf of the form $f(z) = ch(z)$, where $h(z)$ is known, but $c > 0$ could be unknown. Show that the following algorithm returns a random variable Z with pdf f.

- Generate (X, Y) uniformly over the set

$$\mathcal{R} = \{(x, y) : 0 \leqslant x \leqslant \sqrt{h(y/x)}\} .$$

- Return Z = Y/X .

[Hint: consider the coordinate transformation $x = w$, $y = wz$; so $\mathcal{R}$ is transformed to the set $\{(w, z) : 0 \leqslant w \leqslant \sqrt{h(z)}$.]

4. Write a program that generates and displays 100 random vectors that are uniformly distributed within the ellipse

$$5\,x^2 + 21\,x\,y + 25\,y^2 = 9 .$$

Chapter 3

Probability Distributions

This chapter lists some of the major discrete and continuous probability distributions used in Monte Carlo simulation, along with specific algorithms for random variable generation.

3.1 Discrete Distributions

We list various discrete distributions in alphabetical order. Recall that a discrete distribution is completely specified by its discrete pdf.

3.1.1 Bernoulli Distribution

The pdf of the **Bernoulli** distribution is given by

$$f(x;p) = p^x (1-p)^{1-x}, \quad x \in \{0,1\} ,$$

where $p \in [0,1]$ is the **success parameter**. We write the distribution as $\mathsf{Ber}(p)$.

The Bernoulli distribution is used to describe experiments with only two outcomes: 1 (success) or 0 (failure). Such an experiment is called a **Bernoulli trial**. A sequence of iid Bernoulli random variables, $X_1, X_2, \ldots \sim_{\text{iid}} \mathsf{Ber}(p)$, is called a **Bernoulli process**. Such a process is a model for the random experiment where a biased coin is tossed repeatedly. The inverse-transform method leads to the following generation algorithm.

Algorithm 3.1 ($\mathsf{Ber}(p)$ Generator)

1. *Generate* $U \sim \mathsf{U}(0,1)$.
2. *If* $U \leqslant p$, *return* $X = 1$; *otherwise, return* $X = 0$.

Example 3.1 (Bernoulli Generation) The following MATLAB code generates one hundred $\mathsf{Ber}(0.25)$ random variables, and plots a bar graph of the binary data.

```
X = (rand(1,100) <= 0.25);   bar(X)
```

3.1.2 Binomial Distribution

The pdf of the **binomial** distribution is given by

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

where $0 \leqslant p \leqslant 1$. We write the distribution as $\mathsf{Bin}(n,p)$. The binomial distribution is used to describe the total number of successes in a sequence of n independent Bernoulli trials. That is, a $\mathsf{Bin}(n,p)$-distributed random variable X can be written as the sum $X = B_1 + \cdots + B_n$ of independent $\mathsf{Ber}(p)$ random variables $\{B_i\}$. Examples of the graph of the pdf are given in Figure 3.1.

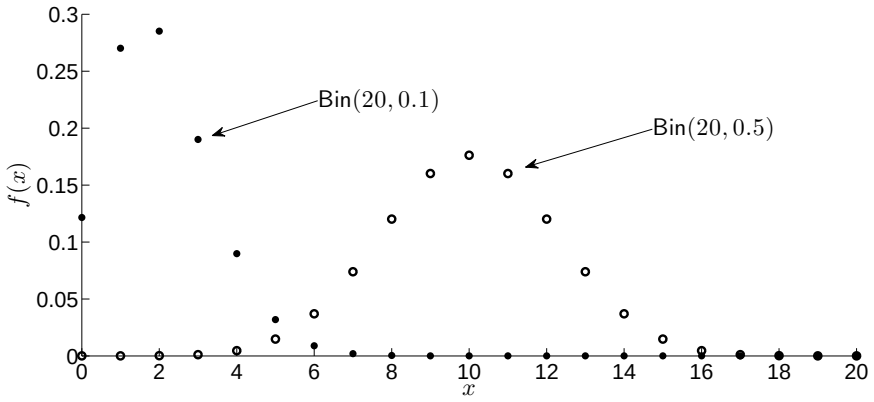


Figure 3.1: The pdfs of the $\mathsf{Bin}(20, 0.1)$ (solid dot) and $\mathsf{Bin}(20, 0.5)$ (circle) distributions.

The fact that binomial random variables can be viewed as sums of Bernoulli random variables leads to the following generation algorithm.

Algorithm 3.2 ($\mathsf{Bin}(n,p)$ Generator)

1. *Generate* $X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathsf{Ber}(p)$.
2. *Return* $X = \sum_{i=1}^{n} X_i$.

Alternative methods should be used for large n.

3.1.3 Geometric Distribution

The pdf of the **geometric** distribution is given by

$$f(x;p) = (1-p)^{x-1} p, \quad x = 1, 2, 3, \ldots \tag{3.1}$$

where $0 \leqslant p \leqslant 1$. We write the distribution as $\mathsf{Geom}(p)$. The geometric distribution is used to describe the time of first success in an infinite sequence of independent Bernoulli trials with success probability p. Examples of the graph of the pdf are given in Figure 3.2.

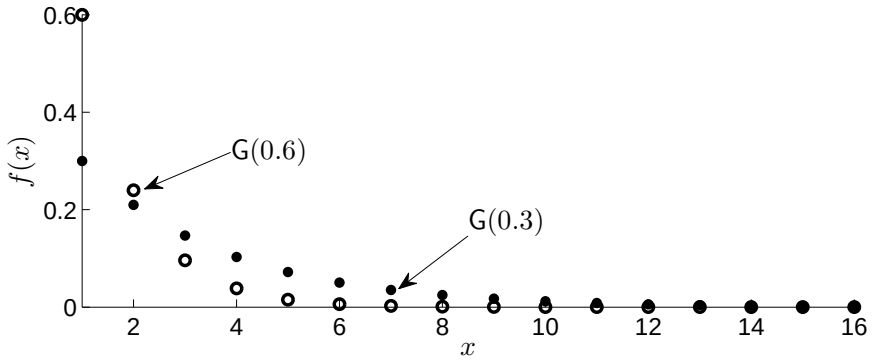


Figure 3.2: The pdfs of the Geom(0.3) (solid dot) and Geom(0.6) (circle) distributions.

Let $Y \sim \mathsf{Exp}(\lambda)$, with $\lambda = -\ln(1-p)$. Then, $\lceil Y \rceil \sim \mathsf{Geom}(p)$. This gives the following generator.

Algorithm 3.3 (Geom(p) Generator (I))

1. *Generate* $Y \sim \mathsf{Exp}(-\ln(1-p))$.
2. *Output* $X = \lceil Y \rceil$.

3.1.4 Poisson Distribution

The pdf of the **Poisson** distribution is given by

$$f(x;\lambda) = \frac{\lambda^x}{x!}\,\mathrm{e}^{-\lambda}, \quad x = 0, 1, 2, \ldots,$$

where $\lambda > 0$ is the **rate** parameter. We write the distribution as $\mathsf{Poi}(\lambda)$. Examples of the graph of the pdf are given in Figure 3.3.

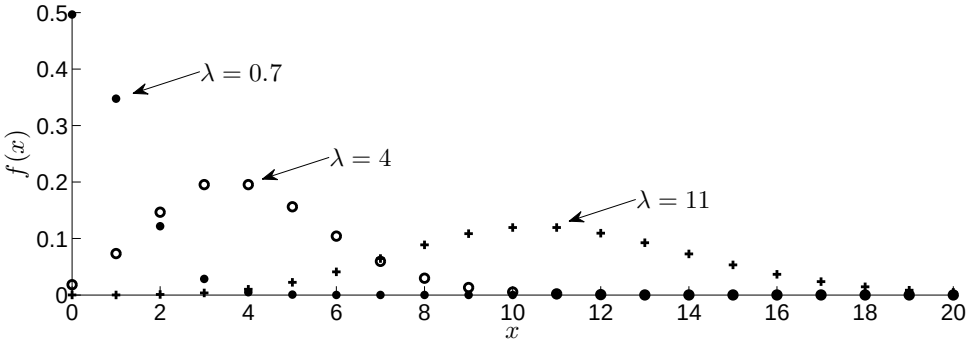


Figure 3.3: The pdfs of the Poi(0.7) (solid dot), Poi(4) (circle), and Poi(11) (plus) distributions.

The Poisson distribution is often used to model the number of arrivals of some sort during a fixed period of time. The Poisson distribution is closely related to the exponential distribution via the **Poisson process**; see Section 4.4. ☞ 69

From this point of view the following is easy to prove: Let $\{Y_i\} \overset{\text{iid}}{\sim} \mathsf{Exp}(\lambda)$. Then,

$$X = \max\left\{n : \sum_{j=1}^{n} Y_j \leqslant 1\right\} \sim \mathsf{Poi}(\lambda) \; . \tag{3.2}$$

That is, the Poisson random variable X can be interpreted as the maximal number of iid exponential variables whose sum does not exceed 1.

Let $\{U_i\} \overset{\text{iid}}{\sim} \mathsf{U}(0,1)$. Rewriting (3.2), we see that

$$\begin{aligned} X &= \max\left\{n : \sum_{j=1}^{n} -\ln U_j \leqslant \lambda\right\} \\ &= \max\left\{n : \ln\left(\prod_{j=1}^{n} U_j\right) \geqslant -\lambda\right\} \\ &= \max\left\{n : \prod_{j=1}^{n} U_j \geqslant \mathrm{e}^{-\lambda}\right\} \end{aligned} \tag{3.3}$$

has a $\mathsf{Poi}(\lambda)$ distribution. This leads to the following algorithm.

Algorithm 3.4 ($\mathsf{Poi}(\lambda)$ Generator)

1. *Set $n = 1$ and $a = 1$.*
2. *Generate $U_n \sim \mathsf{U}(0,1)$ and set $a = a\,U_n$.*
3. *If $a \geqslant \mathrm{e}^{-\lambda}$, set $n = n + 1$ and go to Step 2.*
4. *Otherwise, return $X = n - 1$ as a random variable from $\mathsf{Poi}(\lambda)$.*

For large λ alternative generation methods should be used.

3.1.5 Uniform Distribution (Discrete Case)

The **discrete uniform** distribution has pdf

$$f(x; a, b) = \frac{1}{b - a + 1}, \quad x \in \{a, \ldots, b\} \; ,$$

where $a, b \in \mathbb{Z}$, $b \geqslant a$ are parameters. The discrete uniform distribution is used as a model for choosing a random element from $\{a, \ldots, b\}$ such that each element is equally likely to be drawn. We denote this distribution by $\mathsf{DU}(a, b)$.

Drawing from a discrete uniform distribution on $\{a, \ldots, b\}$, where a and b are integers, is carried out via a simple table lookup method.

Algorithm 3.5 ($\mathsf{DU}(a, b)$ Generator)

Draw $U \sim \mathsf{U}(0,1)$ and output $X = \lfloor a + (b + 1 - a)U \rfloor$.

3.2 Continuous Distributions

We list various continuous distributions in alphabetical order. Recall that an absolutely continuous distribution is completely specified by its pdf.

3.2.1 Beta Distribution

The **beta** distribution has pdf

$$f(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}, \quad x \in [0,1] ,$$

where $\alpha > 0$ and $\beta > 0$ are called **shape** parameters and B is the *beta function*:

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\,\Gamma(\beta)}{\Gamma(\alpha+\beta)} .$$

We write the distribution as $\mathsf{Beta}(\alpha,\beta)$. The dependence of the beta distribution on its shape parameters is illustrated in Figure 3.4.

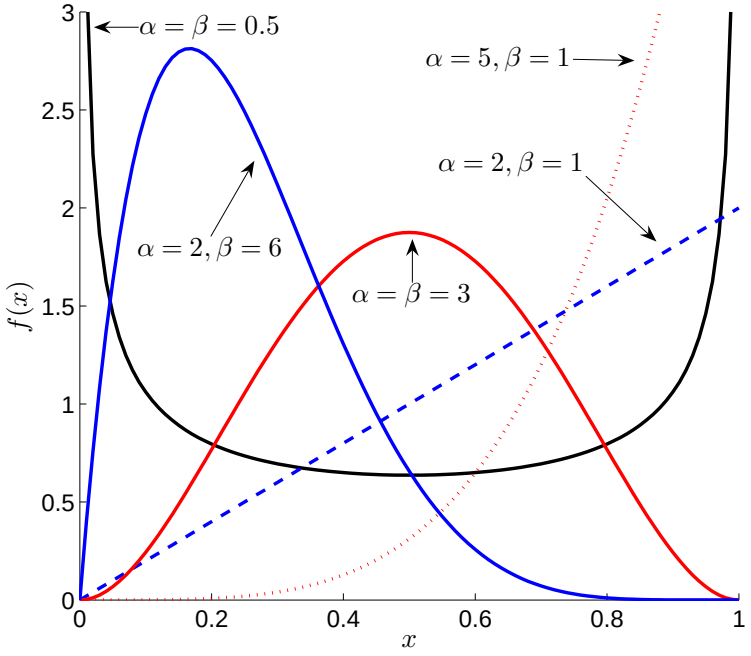


Figure 3.4: Various pdfs of the beta distribution.

A fundamental relation between the gamma and beta distribution is the following: Let $X \sim \mathsf{Gamma}(\alpha,\theta)$ be independent of $Y \sim \mathsf{Gamma}(\beta,\theta)$. Then,

$$\frac{X}{X+Y} \sim \mathsf{Beta}(\alpha,\beta) .$$

More generally, suppose $X_k \sim \mathsf{Gamma}(\alpha_k, 1)$, $k = 1, \ldots, n$, independently. Then, the random variables

$$Y_k = \frac{X_1 + \cdots + X_k}{X_1 + \cdots + X_{k+1}}, \quad k = 1, \ldots, n-1 \,,$$

and $S_n = X_1 + \cdots + X_n$ are independent. Moreover,

$$Y_k \sim \mathsf{Beta}(\alpha_1 + \cdots + \alpha_k, \alpha_{k+1}) \quad \text{and} \quad S_n \sim \mathsf{Gamma}(\alpha_1 + \cdots + \alpha_n, 1) \,.$$

This yields the following generation algorithm.

Algorithm 3.6 (Beta(α, β) Generator)

1. *Generate independently* $Y_1 \sim \mathsf{Gamma}(\alpha, 1)$ *and* $Y_2 \sim \mathsf{Gamma}(\beta, 1)$.
2. *Return* $X = Y_1/(Y_1 + Y_2)$ *as a random variable from* $\mathsf{Beta}(\alpha, \beta)$.

3.2.2 Cauchy Distribution

The **Cauchy** distribution has pdf

$$f(x) = \frac{1}{\pi} \frac{1}{1 + x^2}, \quad x \in \mathbb{R} \,.$$

The graph of the pdf of the $\mathsf{Cauchy}(0, 1)$ distribution is given in Figure 3.5.

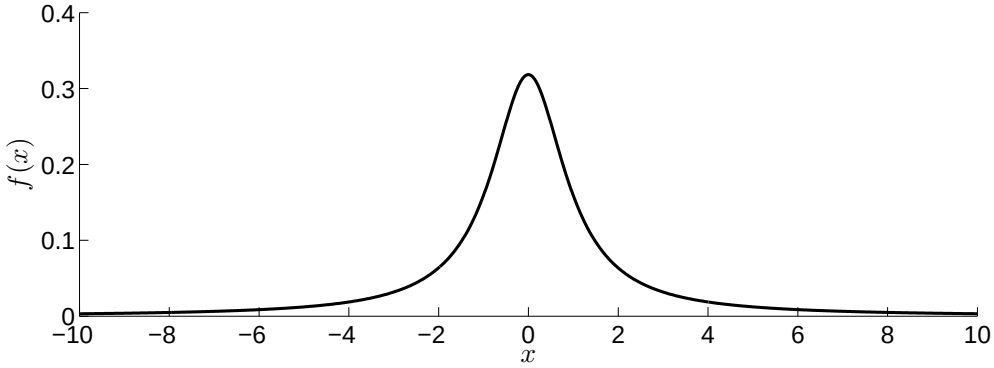


Figure 3.5: The pdf of the Cauchy distribution.

The following algorithm is a direct consequence of the inverse-transform method and the fact that $\cot(\pi x) = \tan\left(\pi x - \frac{\pi}{2}\right)$.

Algorithm 3.7 (Cauchy Generator)

Draw $U \sim \mathsf{U}(0, 1)$ *and output* $X = \cot(\pi U)$ *(or* $X = \tan(\pi U - \pi/2)$*).*

Another simple algorithm is:

Algorithm 3.8 (Cauchy Generator via Ratio of Normals)

1. *Generate* $Y_1, Y_2 \overset{\text{iid}}{\sim} \mathsf{N}(0, 1)$.
2. *Return* $X = Y_1/Y_2$.

3.2.3 Exponential Distribution

The **exponential** distribution has pdf

$$f(x;\lambda) = \lambda \mathrm{e}^{-\lambda x}, \quad x \geqslant 0 \,,$$

where $\lambda > 0$ is the **rate** parameter. We write the distribution as $\mathsf{Exp}(\lambda)$. The exponential distribution can be viewed as a continuous version of the geometric distribution. It plays a central role in the theory and application of Markov jump processes, and in stochastic modeling in general, due to its **memoryless property**: If $X \sim \mathsf{Exp}(\lambda)$, then

☞ 66

$$\mathbb{P}(X > s + t \,|\, X > s) = \mathbb{P}(X > t), \quad s, t \geqslant 0 \,.$$

Graphs of the pdf for various values of λ are given in Figure 3.6.

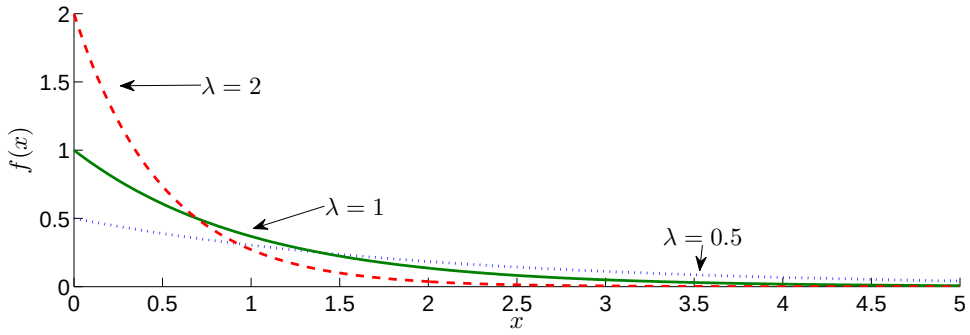


Figure 3.6: Pdfs of the $\mathsf{Exp}(\lambda)$ distribution for various values of λ.

Noting that $U \sim \mathsf{U}(0,1)$ implies $1 - U \sim \mathsf{U}(0,1)$, we obtain the following inverse-transform algorithm.

Algorithm 3.9 ($\mathsf{Exp}(\lambda)$ Generator)

Draw $U \sim \mathsf{U}(0,1)$ *and output* $X = -\frac{1}{\lambda} \ln U$.

3.2.4 Gamma Distribution

The **gamma** distribution has pdf

$$f(x;\alpha,\lambda) = \frac{\lambda^\alpha x^{\alpha-1} \mathrm{e}^{-\lambda x}}{\Gamma(\alpha)}, \quad x \geqslant 0 \,, \tag{3.4}$$

where $\alpha > 0$ is called the **shape** parameter and $\lambda > 0$ the **scale** parameter. In the formula for the pdf, Γ is the *gamma function*. We write the distribution as $\mathsf{Gamma}(\alpha, \lambda)$.

An important special case is the $\mathsf{Gamma}(n/2, 1/2)$ distribution with $n \in \{1, 2, \ldots\}$, which is called a **chi-square** distribution; the parameter n is then referred to as the **number of degrees of freedom**. The distribution is written

as χ^2_n. A graph of the pdf of the χ^2_n distribution, for various n, is given in Figure 3.7.

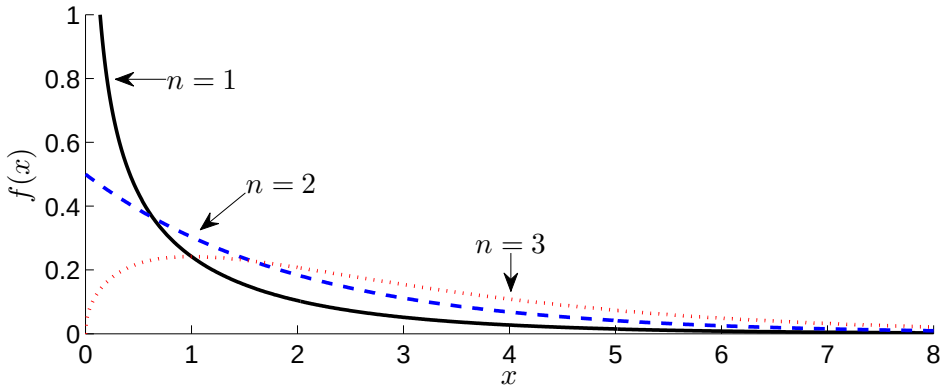


Figure 3.7: Pdfs of the χ^2_n distribution for various degrees of freedom n.

Since $\mathsf{Gamma}(\alpha, \lambda)$ is a scale family, it suffices to only give algorithms for generating random variables $X \sim \mathsf{Gamma}(\alpha, 1)$, because $X/\lambda \sim \mathsf{Gamma}(\alpha, \lambda)$. Since the cdf of the gamma distribution does not generally exist in explicit form, the inverse-transform method cannot always be applied to generate random variables from this distribution. Thus, alternative methods are called for. The following algorithm, by Marsaglia and Tsang provides a highly efficient acceptance–rejection method for generating $\mathsf{Gamma}(\alpha, 1)$ random variables with $\alpha \geqslant 1$.

Algorithm 3.10 ($\mathsf{Gamma}(\alpha, 1)$ Generator for $\alpha \geqslant 1$)

1. *Set* $d = \alpha - 1/3$ *and* $c = 1/\sqrt{9\,d}$.
2. *Generate* $Z \sim \mathsf{N}(0,1)$ *and* $U \sim \mathsf{U}(0,1)$ *independently.*
3. *If* $Z > -1/c$ *and* $\ln U < \frac{1}{2}Z^2 + d - d\,V + d \ln V$, *where* $V = (1 + c\,Z)^3$, *return* $X = d\,V$; *otherwise, go back to Step 2.*

For the case $\alpha < 1$ one can use the fact that if $X \sim \mathsf{Gamma}(1 + \alpha, 1)$, and $U \sim \mathsf{U}(0,1)$ are independent, then $XU^{1/a} \sim \mathsf{Gamma}(\alpha, 1)$. Alternatively, one can use the following algorithm by Best:

Algorithm 3.11 ($\mathsf{Gamma}(\alpha, 1)$ Generator for $\alpha < 1$)

1. *Set* $d = 0.07 + 0.75\sqrt{1 - \alpha}$ *and* $b = 1 + \mathrm{e}^{-d}\alpha/d$.
2. *Generate* $U_1, U_2 \overset{\text{iid}}{\sim} \mathsf{U}(0,1)$ *and set* $V = b\,U_1$.
3. *If* $V \leqslant 1$, *then set* $X = d\,V^{1/\alpha}$. *Check whether* $U_2 \leqslant (2 - X)/(2 + X)$. *If true, return* X; *otherwise, check whether* $U_2 \leqslant \mathrm{e}^{-X}$. *If true, return* X; *otherwise, go back to Step 2.*

If $V > 1$, then set $X = -\ln(d(b-V)/\alpha)$ and $Y = X/d$. Check whether $U_2(\alpha + y(1-\alpha)) \leqslant 1$. If true, return X; otherwise, check if $U_2 < Y^{\alpha-1}$. If true, return X; otherwise, go back to Step 2.

3.2.5 Normal Distribution

The standard **normal** or standard **Gaussian** distribution has pdf

$$f(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2}, \quad x \in \mathbb{R}\,.$$

The corresponding location–scale family of pdfs is therefore

$$f(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \; x \in \mathbb{R}\,. \tag{3.5}$$

We write the distribution as $\mathsf{N}(\mu, \sigma^2)$. We denote the pdf and cdf of the $\mathsf{N}(0,1)$ distribution as φ and Φ, respectively. Here, $\Phi(x) = \int_{-\infty}^{x} \varphi(t)\,\mathrm{d}t = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)$, where $\mathrm{erf}(x)$ is the *error function*.

The normal distribution plays a central role in statistics and arises naturally as the limit of the sum of iid random variables via the central limit theorem. Its crucial property is that any affine combination of independent normal random variables is again normal. In Figure 3.8 the probability densities for three different normal distributions are depicted.

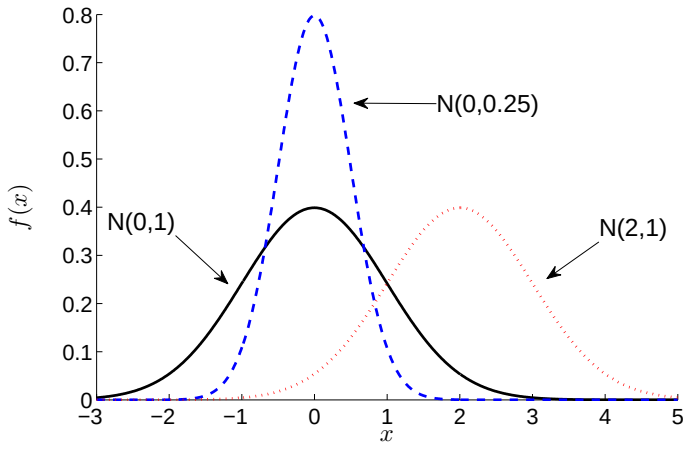


Figure 3.8: Pdfs of the normal distribution for various values of the parameters.

Since $\mathsf{N}(\mu, \sigma)$ forms a location–scale family, we only consider generation from $\mathsf{N}(0,1)$. The most prominent application of the polar method (see Section 2.1.2.6) lies in the generation of standard normal random variables, leading to the celebrated Box–Muller method. ☞ 34

Algorithm 3.12 (N(0, 1) Generator, Box–Muller Approach)

1. *Generate* $U_1, U_2 \overset{\text{iid}}{\sim} \mathsf{U}(0,1)$.
2. *Return two independent standard normal variables,* X *and* Y*, via*

$$\begin{aligned} X &= \sqrt{-2\ln U_1}\,\cos(2\pi U_2)\ , \\ Y &= \sqrt{-2\ln U_1}\,\sin(2\pi U_2)\ . \end{aligned} \tag{3.6}$$

Finally, the following algorithm uses acceptance–rejection with an exponential proposal distribution. This gives a probability of acceptance of $\sqrt{\pi/(2\mathrm{e})} \approx 0.76$. The theory behind it is given in Example 2.12.

☞ 39

Algorithm 3.13 (N(0, 1) Generator, Acceptance–Rejection from Exp(1))

1. *Generate* $X \sim \mathsf{Exp}(1)$ *and* $U' \sim \mathsf{U}(0,1)$*, independently.*
2. *If* $U' \leqslant \mathrm{e}^{-(X-1)^2/2}$*, generate* $U \sim \mathsf{U}(0,1)$ *and output* $Z = (1 - 2\,\mathrm{I}_{\{U \leqslant 1/2\}})\,X$*; otherwise, repeat from Step 1.*

3.2.6 Uniform Distribution (Continuous Case)

The **uniform** distribution on the interval $[a, b]$ has pdf

$$f(x; a, b) = \frac{1}{b-a}, \quad a \leqslant x \leqslant b\ .$$

We write the distribution as $\mathsf{U}[a, b]$. A graph of the pdf is given in Figure 3.9.

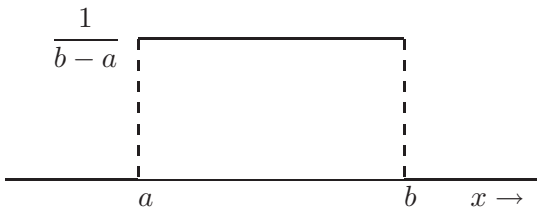


Figure 3.9: The pdf of the uniform distribution on $[a, b]$.

The uniform distribution is used as a model for choosing a point randomly from the interval $[a, b]$, such that each point is equally likely to be drawn. The uniform distribution on an arbitrary Borel set B in $\mathbb{R}^n$ with non-zero Lebesgue measure (for example, area, volume) $|B|$ is defined similarly: its pdf is constant, taking value $1/|B|$ on B and 0 otherwise. We write $\mathsf{U}(B)$ or simply $\mathsf{U}B$. The $\mathsf{U}[a, b]$ distribution is a location–scale family, as $Z \sim \mathsf{U}[a, b]$ has the same distribution as $a + (b - a)X$, with $X \sim \mathsf{U}[0, 1]$.

The generation of $\mathsf{U}(0, 1)$ random variables, crucial for any Monte Carlo method, is discussed in detail in Chapter 1. $\mathsf{U}(a, b)$ random variable generation follows immediately from the inverse-transform method.

☞ 9

Algorithm 3.14 (U(a, b) Generation)

Generate $U \sim \mathsf{U}(0,1)$ *and return* $X = a + (b - a)U$.

3.3 Multivariate Distributions

3.3.1 Dirichlet Distribution

The **standard Dirichlet** (or **type I Dirichlet**) distribution has pdf

$$f(\mathbf{x};\boldsymbol{\alpha}) = \frac{\Gamma\big(\sum_{i=1}^{n+1}\alpha_i\big)}{\prod_{i=1}^{n+1}\Gamma(\alpha_i)}\prod_{i=1}^{n} x_i^{\alpha_i-1}\left(1-\sum_{i=1}^{n} x_i\right)^{\alpha_{n+1}-1}, \quad x_i \geqslant 0, i=1,\ldots,n, \sum_{i=1}^{n} x_i \leqslant 1\ ,$$

where $\alpha_i > 0$, $i = 1,\ldots,n+1$ are **shape** parameters. We write this distribution as $\mathsf{Dirichlet}(\alpha_1,\ldots,\alpha_{n+1})$ or $\mathsf{Dirichlet}(\boldsymbol{\alpha})$, with $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_{n+1})^\top$. The Dirichlet distribution can be regarded as a multivariate generalization of the beta distribution (see Section 3.2.1), in the sense that each marginal X_k has a beta distribution. A graph of the pdf for the two-dimensional case is given in Figure 3.10.

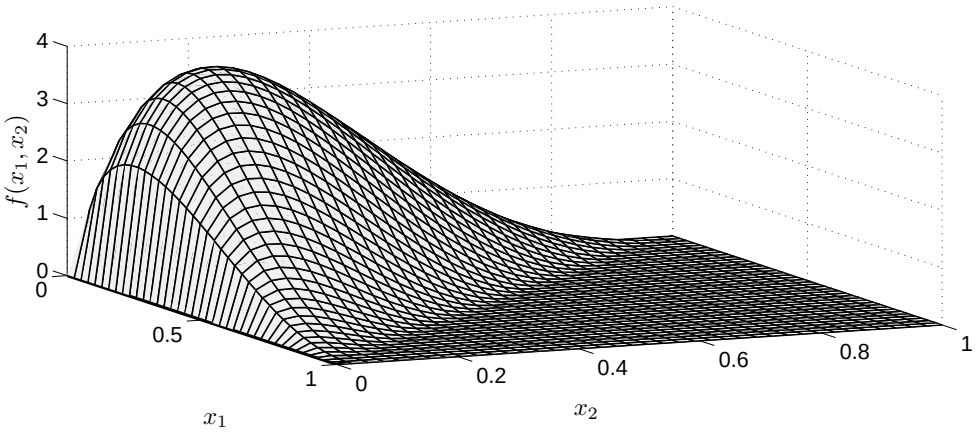


Figure 3.10: The Dirichlet pdf in two dimensions, with parameter vector $\boldsymbol{\alpha} = (1.5, 1.5, 3)^\top$.

The fundamental relation between the Dirichlet and the gamma distribution given is the following:

Let $Y_1,\ldots,Y_{n+1}$ be independent random variables with $Y_k \sim \mathsf{Gamma}(\alpha_k, 1)$, $k = 1,\ldots,n+1$, and define

$$X_k = \frac{Y_k}{\sum_{i=1}^{n+1} Y_i}, \quad k = 1,\ldots,n\ .$$

Then, $\mathbf{X} = (X_1,\ldots,X_n) \sim \mathsf{Dirichlet}(\boldsymbol{\alpha})$.

This provides the following generation method.

Algorithm 3.15 (Dirichlet($\boldsymbol{\alpha}$) Generator)

1. *Generate* $Y_k \sim \mathsf{Gamma}(\alpha_k, 1)$, $k = 1,\ldots,n+1$ *independently.*
2. *Output* $\mathbf{X} = (X_1,\ldots,X_n)$, *where*

$$X_k = \frac{Y_k}{\sum_{i=1}^{n+1} Y_i}, \quad k = 1,\ldots,n\ .$$

3.3.2 Multivariate Normal Distribution

The **standard multivariate normal** or **standard multivariate Gaussian** distribution in n dimensions has pdf

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n}} \mathrm{e}^{-\frac{1}{2}\,\mathbf{x}^\top \mathbf{x}}, \quad \mathbf{x} \in \mathbb{R}^n \,. \tag{3.7}$$

All marginals of $\mathbf{X} = (X_1, \ldots, X_n)^\top$ are iid standard normal random variables.

Suppose that $\mathbf{Z}$ has an m-dimensional standard normal distribution. If A is an $n \times m$ matrix and $\boldsymbol{\mu}$ is an $n \times 1$ vector, then the affine transformation

$$\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Z}$$

is said to have a **multivariate normal** or **multivariate Gaussian** distribution with **mean vector** $\boldsymbol{\mu}$ and **covariance matrix** $\Sigma = AA^\top$. We write the distribution as $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$.

The covariance matrix Σ is always symmetric and positive semidefinite. When A is of full rank (that is, $\mathrm{rank}(A) = \min\{m, n\}$) the covariance matrix Σ is positive definite. In this case Σ has an inverse and the distribution of $\mathbf{X}$ has pdf

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \mathrm{e}^{-\frac{1}{2}\,(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})}, \quad \mathbf{x} \in \mathbb{R}^n \,. \tag{3.8}$$

The multivariate normal distribution is a natural extension of the normal distribution (see Section 3.2.5) and plays a correspondingly important role in multivariate statistics. This multidimensional counterpart also has the property that any affine combination of independent multivariate normal random variables is again multivariate normal. A graph of the standard normal pdf for the two-dimensional case is given in Figure 3.11.

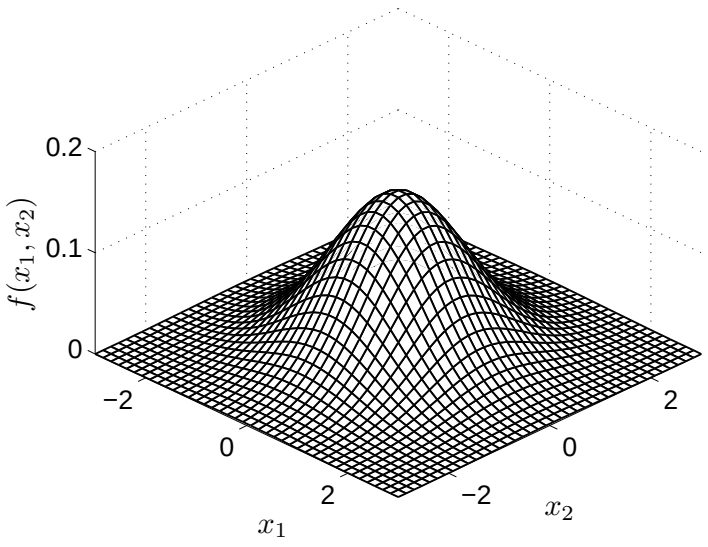


Figure 3.11: The standard multivariate normal pdf in two dimensions.

Some important properties are:

1. *Affine combinations*: Let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_r$ be independent m_i-dimensional normal variables, with $\mathbf{X}_i \sim \mathsf{N}(\boldsymbol{\mu}_i, \Sigma_i)$, $i = 1, \ldots, r$. Let $\mathbf{a}$ be an $n \times 1$ vector and let each A_i be an $n \times m_i$ matrix for $i = 1, \ldots, r$. Then,

$$\mathbf{a} + \sum_{i=1}^{r} A_i \, \mathbf{X}_i \sim \mathsf{N}\left(\mathbf{a} + \sum_{i=1}^{r} A_i \, \boldsymbol{\mu}_i, \; \sum_{i=1}^{r} A_i \, \Sigma_i \, A_i^\top \right).$$

   In other words, any affine combination of independent multivariate normal random variables is again multivariate normal.

2. *Standardization (whitening)*: A particular case of the affine combinations property is the following. Suppose $\mathbf{X} \sim \mathsf{N}(\boldsymbol{\mu}, \Sigma)$ is an n-dimensional normal random variable with $\det(\Sigma) > 0$. Let A be the Cholesky factor of the matrix Σ. That is, A is an $n \times n$ lower triangular matrix of the form

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \tag{3.9}$$

   and such that $\Sigma = AA^\top$. It follows that

$$A^{-1}(\mathbf{X} - \boldsymbol{\mu}) \sim \mathsf{N}(\mathbf{0}, I).$$

3. *Marginal distributions*: Let $\mathbf{X}$ be an n-dimensional normal variable, with $\mathbf{X} \sim \mathsf{N}(\boldsymbol{\mu}, \Sigma)$. Separate the vector $\mathbf{X}$ into a part of size p and one of size $q = n - p$ and, similarly, partition the mean vector and covariance matrix:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_p \\ \mathbf{X}_q \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_p \\ \boldsymbol{\mu}_q \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_p & \Sigma_r \\ \Sigma_r^\top & \Sigma_q \end{pmatrix}, \tag{3.10}$$

   where Σ_p is the upper left $p \times p$ corner of Σ, Σ_q is the lower right $q \times q$ corner of Σ, and Σ_r is the $p \times q$ upper right block of Σ. Then, the distributions of the marginal vectors $\mathbf{X}_p$ and $\mathbf{X}_q$ are also multivariate normal, with $\mathbf{X}_p \sim \mathsf{N}(\boldsymbol{\mu}_p, \Sigma_p)$ and $\mathbf{X}_q \sim \mathsf{N}(\boldsymbol{\mu}_q, \Sigma_q)$.

   Note that an arbitrary selection of p and q elements can be achieved by first performing a linear transformation $\mathbf{Z} = A\mathbf{X}$, where A is the $n \times n$ *permutation matrix* that appropriately rearranges the elements in $\mathbf{X}$.

4. *Conditional distributions*: Suppose we again have an n-dimensional vector $\mathbf{X} \sim \mathsf{N}(\boldsymbol{\mu}, \Sigma)$, partitioned as for the marginal distribution property above, but with $\det(\Sigma) > 0$. Then, we have the conditional distributions

$$(\mathbf{X}_p \,|\, \mathbf{X}_q = \mathbf{x}_q) \sim \mathsf{N}(\boldsymbol{\mu}_p + \Sigma_r \Sigma_q^{-1}(\mathbf{x}_q - \boldsymbol{\mu}_q), \; \Sigma_p - \Sigma_r \Sigma_q^{-1} \Sigma_r^\top),$$

   and

$$(\mathbf{X}_q \,|\, \mathbf{X}_p = \mathbf{x}_p) \sim \mathsf{N}(\boldsymbol{\mu}_q + \Sigma_r^\top \Sigma_p^{-1}(\mathbf{x}_p - \boldsymbol{\mu}_p), \; \Sigma_q - \Sigma_r^\top \Sigma_p^{-1} \Sigma_r).$$

As with the marginals property, arbitrary conditioning can be achieved by first permuting the elements of $\mathbf{X}$ by way of an affine transformation using a permutation matrix.

Property 2 is the key to generating a multivariate normal random vector $\mathbf{X} \sim \mathsf{N}(\boldsymbol{\mu}, \Sigma)$, and leads to the following algorithm.

Algorithm 3.16 ($\mathsf{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ Generator)

1. *Derive the Cholesky decomposition* $\Sigma = AA^\top$.
2. *Generate* $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$ *and let* $\mathbf{Z} = (Z_1, \ldots, Z_n)^\top$.
3. *Output* $\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Z}$.

3.3.3 Multivariate Student's t Distribution

The **multivariate Student's t** distribution in n dimensions has pdf

$$f(\mathbf{x}; \nu) = \frac{\Gamma\left(\frac{\nu+n}{2}\right)}{(\pi\nu)^{n/2}\,\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{1}{\nu}\mathbf{x}^\top\mathbf{x}\right)^{-\frac{\nu+n}{2}}, \quad \mathbf{x} \in \mathbb{R}^n, \tag{3.11}$$

where $\nu \geqslant 0$ is the **degrees of freedom** or **shape** parameter. We write the distribution as $\mathbf{t}_\nu$. Suppose $\mathbf{Y} = (Y_1, \ldots, Y_m)^\top$ has an m-dimensional $\mathbf{t}_\nu$ distribution. If A is an $n \times m$ matrix and $\boldsymbol{\mu}$ is an $n \times 1$ vector, then the affine transformation

$$\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Y}$$

is said to have a **multivariate Student's** or **multivariate t** distribution with **mean vector $\boldsymbol{\mu}$** and **scale matrix** $\Sigma = AA^\top$. We write the distribution as $\mathbf{t}_\nu(\boldsymbol{\mu}, \Sigma)$. The multivariate t distribution is a *radially symmetric* extension of the univariate t distribution and plays an important role as a proposal distribution in MCMC algorithms and Bayesian statistical modeling; see Example 5.1.

☞ 89

The key relation between the multivariate t distribution and the multivariate normal distribution is the following:

Let $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)$ and $S \sim \mathsf{Gamma}(\nu/2, 1/2)$ be independent. Then,

$$\mathbf{X} = \sqrt{\frac{\nu}{S}}\,\mathbf{Z} \sim \mathbf{t}_\nu\ .$$

This leads to the next generation algorithm.

Algorithm 3.17 ($\mathbf{t}_\nu(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ Generator)

1. *Draw the random column vector* $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)$.
2. *Draw* $S \sim \mathsf{Gamma}\left(\frac{\nu}{2}, \frac{1}{2}\right) \equiv \chi^2_\nu$.
3. *Compute* $\mathbf{Y} = \sqrt{\frac{\nu}{S}}\,\mathbf{Z}$.
4. *Return* $\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Y}$*, where* A *is the Cholesky factor of* Σ*, so that* $AA^\top = \Sigma$.

3.4 Exercises

1. Construct generation algorithms for the following distributions:

(a) The $\mathsf{Weib}(\alpha, \lambda)$ distribution, with cdf $F(x) = 1 - \mathrm{e}^{-(\lambda x)^\alpha}, x \geqslant 0$, where $\lambda > 0$ and $\alpha > 0$.

(b) The $\mathsf{Pareto}(\alpha, \lambda)$ distribution, with pdf $f(x) = \alpha\lambda(1 + \lambda x)^{-(\alpha+1)}, x \geqslant 0$, where $\lambda > 0$ and $\alpha > 0$.

2. Write a MATLAB program for drawing random samples from the following distributions (do not use the standard MATLAB random number generators, but write your own, based on the algorithms in this chapter):

1. $\mathsf{Ber}(p)$ (tossing a coin).
2. The discrete uniform distribution on $\{1, \ldots, 6\}$ (tossing a fair die).
3. $\mathsf{Bin}(10, p)$, where $p \in (0, 1)$.
4. $\mathsf{Geom}(p)$, where $p \in (0, 1)$.
5. $\mathsf{Exp}(\lambda)$, where $\lambda > 0$.
6. $\mathsf{Beta}(\alpha, 1)$, where $\alpha > 0$.
7. The standard normal distribution (use the Box-Muller algorithm).
8. The $\mathsf{N}(\mu, \sigma^2)$ distribution.
9. The multivariate $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$ distribution.

3. Implement Algorithm 3.10 for the gamma distribution. Verify that (for various values of α and λ) the algorithm draws from the correct distribution by generating $N = 10^5$ samples and comparing the true pdf with the estimated density obtained via the kernel density estimation program `kde.m` from
`http://www.mathworks.com/matlabcentral/fileexchange/14034`

4. Apply the inverse-transform method to generate a random variable from the *extreme value distribution*, which has cdf

$$F(x) = 1 - \mathrm{e}^{-\exp(\frac{x-\mu}{\sigma})}, \quad -\infty < x < \infty, \quad (\sigma > 0).$$

5. If X and Y are independent standard normal random variables, then $Z = X/Y$ has a Cauchy distribution. Show this. (Hint: first show that if U and $V > 0$ are continuous random variables with joint pdf $f_{U,V}$, then the pdf of $W = U/V$ is given by $f_W(w) = \int_0^\infty f_{U,V}(w\,v, v)\, v\, \mathrm{d}v$.)

Chapter 4

Random Process Generation

This chapter lists a selection of the main random processes used in Monte Carlo simulation, along with their generation algorithms.

4.1 Gaussian Processes

A real-valued stochastic process $\{\widetilde{X}_t, t \in \mathscr{T}\}$ is said to be a **Gaussian process** if all its finite-dimensional distributions are Gaussian (normal); that is, if $\mathbf{X} = (X_1, \ldots, X_n) = (\widetilde{X}_{t_1}, \ldots, \widetilde{X}_{t_n})^\top$ has a multivariate Gaussian distribution for any choice of n and $t_1, \ldots, t_n \in \mathscr{T}$, or equivalently, if any linear combination $\sum_{i=1}^n b_i \widetilde{X}_{t_i}$ has a normal distribution. ☞ 56

The probability distribution of a Gaussian process is determined completely by its **expectation function**

$$\widetilde{\mu}_t = \mathbb{E}\widetilde{X}_t, \quad t \in \mathscr{T}$$

and **covariance function**

$$\widetilde{\Sigma}_{s,t} = \mathrm{Cov}(\widetilde{X}_s, \widetilde{X}_t), \quad s, t \in \mathscr{T} \ .$$

A **zero-mean** Gaussian process is one for which $\widetilde{\mu}_t = 0$ for all t.

Gaussian processes can be thought of as generalizations of Gaussian random vectors. To generate a realization of a Gaussian process with expectation function $(\widetilde{\mu}_t)$ and covariance function $(\widetilde{\Sigma}_{s,t})$ at times $t_1, \ldots, t_n$ we can simply sample a multivariate normal random vector $\mathbf{X} = (X_1, \ldots, X_n)^\top = (\widetilde{X}_{t_1}, \ldots, \widetilde{X}_{t_n})^\top$. As such, the fundamental generation method is the same as given in Algorithm 3.16. ☞ 58

Algorithm 4.1 (Gaussian Process Generator)

1. *Construct the mean vector* $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)^\top$ *and covariance matrix* $\Sigma = (\Sigma_{ij})$ *by setting* $\mu_i = \widetilde{\mu}_{t_i}, i = 1, \ldots, n$ *and* $\Sigma_{ij} = \widetilde{\Sigma}_{t_i, t_j}$, $i, j = 1, \ldots, n$.
2. *Derive the Cholesky decomposition* $\Sigma = AA^\top$.
3. *Generate* $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$. *Let* $\mathbf{Z} = (Z_1, \ldots, Z_n)^\top$.
4. *Output* $\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Z}$.

4.1.1 Markovian Gaussian Processes

Let $\{\widetilde{X}_t, t \geqslant 0\}$ be a real-valued *Markovian* Gaussian process. Thus, in addition to being Gaussian, the process also satisfies the **Markov property**:

$$(\widetilde{X}_{t+s} \,|\, \widetilde{X}_u, u \leqslant t) \sim (\widetilde{X}_{t+s} \,|\, \widetilde{X}_t) \quad \text{for all } s, t \geqslant 0 \,.$$

If the mean $(\widetilde{\mu}_t)$ and covariance function $(\widetilde{\Sigma}_{s,t})$ are known, then it is straightforward to generate realizations $(X_1, \ldots, X_n) = (\widetilde{X}_{t_1}, \ldots, \widetilde{X}_{t_n})$ of the process at any selection of times $0 \leqslant t_1 < \ldots < t_n$ by using the *conditional distributions* property of the multivariate normal distribution; see Property 4 on Page 57. Denote the expectation and variance of $X_i = \widetilde{X}_{t_i}$ by μ_i and $\sigma_{i,i}$, respectively and let $\sigma_{i,i+1} = \mathrm{Cov}(X_i, X_{i+1})$, $i = 1, \ldots, n-1$. Then, by the marginal distributions property (Property 3 on Page 57),

$$\begin{pmatrix} X_i \\ X_{i+1} \end{pmatrix} \sim \mathsf{N}\left(\begin{pmatrix} \mu_i \\ \mu_{i+1} \end{pmatrix}, \begin{pmatrix} \sigma_{i,i} & \sigma_{i,i+1} \\ \sigma_{i,i+1} & \sigma_{i+1,i+1} \end{pmatrix} \right) .$$

Hence, by the conditional distributions property we have

$$(X_{i+1} \,|\, X_i = x) \sim \mathsf{N}\left(\mu_{i+1} + \frac{\sigma_{i,i+1}}{\sigma_{i,i}}(x - \mu_i), \quad \sigma_{i+1,i+1} - \frac{\sigma_{i,i+1}^2}{\sigma_{i,i}} \right) .$$

This leads to the following algorithm.

Algorithm 4.2 (Generating a Markovian Gaussian Process)

1. *Draw* $Z \sim \mathsf{N}(0,1)$ *and set* $X_1 = \mu_1 + \sqrt{\sigma_{1,1}}\, Z$.
2. *For* $i = 1, \ldots, n-1$, *draw* $Z \sim \mathsf{N}(0,1)$ *and set*

$$X_{i+1} = \mu_{i+1} + \frac{\sigma_{i+1,i}}{\sigma_{i,i}} (X_i - \mu_i) + \sqrt{\sigma_{i+1,i+1} - \frac{\sigma_{i,i+1}^2}{\sigma_{i,i}}}\; Z \,.$$

The algorithm can be easily generalized to generate multidimensional Markovian Gaussian processes. In particular, let $\{\widetilde{\mathbf{X}}_t, t \geqslant 0\}$ be a d-dimensional Markovian Gaussian process with expectation function $\widetilde{\boldsymbol{\mu}}_t = \mathbb{E}\widetilde{\mathbf{X}}_t$, $t \geqslant 0$, and covariance function $\widetilde{\Sigma}_{s,t} = \mathrm{Cov}(\widetilde{\mathbf{X}}_s, \widetilde{\mathbf{X}}_t)$, $s, t \geqslant 0$. The following algorithm generates realizations of the process at times $0 \leqslant t_1 < \ldots < t_n$.

Algorithm 4.3 (Generating a Multidimensional Markovian Gaussian Process)

1. *Draw* $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)$ *and set* $\widetilde{\mathbf{X}}_{t_1} = \widetilde{\boldsymbol{\mu}}_{t_1} + B\mathbf{Z}$, *where* B *is the (lower-triangular) Cholesky square root matrix of* $\widetilde{\Sigma}_{t_1,t_1}$.
2. *For* $k = 1, \ldots, n-1$,
   (a) *Compute the Cholesky square-root of*

$$\widetilde{\Sigma}_{t_{k+1},t_{k+1}} - \widetilde{\Sigma}_{t_k,t_{k+1}} \widetilde{\Sigma}_{t_k,t_k}^{-1} \widetilde{\Sigma}_{t_k,t_{k+1}} \,,$$

   and denote it C.
   (b) *Draw* $\mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)$ *and set*

$$\widetilde{\mathbf{X}}_{t_{k+1}} = \widetilde{\boldsymbol{\mu}}_{t_{k+1}} + \widetilde{\Sigma}_{t_{k+1},t_k} \widetilde{\Sigma}_{t_k,t_k}^{-1} \left(\mathbf{X}_{t_k} - \widetilde{\boldsymbol{\mu}}_{t_k} \right) + C\,\mathbf{Z} \,.$$

4.2 Markov Chains

A **Markov chain** is a stochastic process $\{X_t, t \in \mathscr{T}\}$ with a countable index set $\mathscr{T} \subset \mathbb{R}$ which satisfies the **Markov property**

$$(X_{t+s} \,|\, X_u, u \leqslant t) \sim (X_{t+s} \,|\, X_t) \; .$$

We discuss here only the main points pertinent to the simulation of such processes. We assume throughout that the index set is $\mathscr{T} = \{0, 1, 2, \ldots\}$.

A direct consequence of the Markov property is that Markov chains can be generated *sequentially*: $X_0, X_1, \ldots$, as expressed in the following generic recipe.

Algorithm 4.4 (Generating a Markov Chain)

1. *Draw X_0 from its distribution. Set $t = 0$.*
2. *Draw X_{t+1} from the conditional distribution of X_{t+1} given X_t.*
3. *Set $t = t + 1$ and repeat from Step 2.*

The conditional distribution of X_{t+1} given X_t can be specified in two common ways as follows.

- The process $\{X_t, t = 0, 1, 2, \ldots\}$ satisfies a recurrence relation

$$X_{t+1} = g(t, X_t, U_t) \; , \quad t = 0, 1, 2 \ldots \; , \tag{4.1}$$

  where g is an easily evaluated function and U_t is an easily generated random variable whose distribution may depend on X_t and t.

- The conditional distribution of X_{t+1} given X_t is known and is easy to sample from.

An important instance of the second case occurs when the Markov chain $\{X_0, X_1, \ldots\}$ has a discrete state space E and is time-homogeneous. Its distribution is then completely specified by the distribution of X_0 (the initial distribution) and the matrix of one-step transition probabilities $P = (p_{ij})$, where

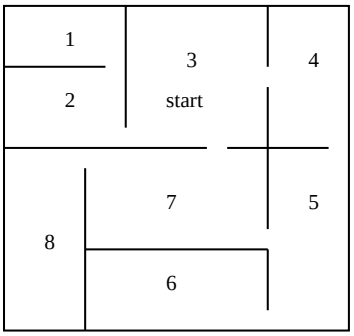
$$p_{ij} = \mathbb{P}(X_{t+1} = j \,|\, X_t = i), \quad i, j \in E \; .$$

The conditional distribution of X_{n+1} given $X_n = i$ is therefore a discrete distribution given by the i-th row of P. This leads to the following specification of Algorithm 4.4.

Algorithm 4.5 (Generating a Time-Homogeneous Markov Chain on a Discrete State Space)

1. *Draw X_0 from the initial distribution. Set $t = 0$.*
2. *Draw X_{t+1} from the discrete distribution corresponding to the X_t-th row of P.*
3. *Set $t = t + 1$ and go to Step 2.*

Example 4.1 (A Markov Chain Maze) At time $t = 0$ a robot is placed in compartment 3 of the maze in Figure 4.1. At each time $t = 1, 2, \ldots$ the robot chooses one of the adjacent compartments with equal probability. Let X_t be the robot's compartment at time t. Then $\{X_t\}$ is a time-homogeneous Markov chain with transition matrix P given below.



$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Figure 4.1: A maze and the corresponding transition matrix.

The following MATLAB program implements Algorithm 4.5. The first 100 values of the process are given in Figure 4.2.

```
%maze.m
n = 101
a = 0.5; b = 1/3;
P = [0, 1, 0, 0, 0, 0, 0, 0; a, 0, a, 0, 0, 0, 0, 0;
     0, b, 0, b, 0, 0, b, 0; 0, 0, a, 0, a, 0, 0, 0;
     0, 0, 0, b, 0, b, b, 0; 0, 0, 0, 0, 1, 0, 0, 0;
     0, 0, b, 0, b, 0, 0, b; 0, 0, 0, 0, 0, 0, 1, 0 ]
x = zeros(1,n);
x(1)= 3;
for t=1:n-1
    x(t+1) = min(find(cumsum(P(x(t),:))> rand));
end
hold on
plot(0:n-1,x,'.')
plot(0:n-1,x)
hold off
```

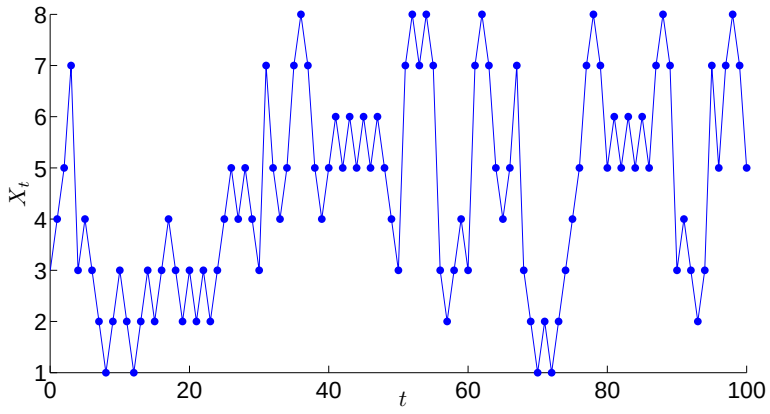


Figure 4.2: Realization of the maze process.

Example 4.2 (Random Walk on an n-Dimensional Hypercube) A typical example of a Markov chain that is specified by a recurrence relation such as (4.1) is the **random walk** process. Here the recurrence is simply

$$X_{t+1} = X_t + U_t \;, \quad t = 1, 2, \ldots \;,$$

where $U_1, U_2, \ldots$ is an iid sequence of random variables from some discrete or continuous distribution.

A similar recurrence can be used to generate a random walk on the set of vertices of the unit n-dimensional hypercube — that is, the set of binary vectors of length n. Denote by $\mathbf{e}_1, \ldots, \mathbf{e}_n$ the unit vectors in $\mathbb{R}^n$. Starting with $\mathbf{X}_0$ somewhere on the unit hypercube, define

$$\mathbf{X}_{t+1} = \mathbf{X}_t + \mathbf{e}_{I_t} \mod 2 \;,$$

where $I_1, I_2, \ldots \overset{\text{iid}}{\sim} \mathsf{DU}(1, \ldots, n)$. Note that $\mathbf{e}_I$ is simply a uniformly chosen unit vector. Thus, the process performs a random walk on the unit hypercube, where at each step the process jumps from one vertex to one of the n adjacent vertices with equal probability. Since $\mathbf{X}_{t+1}$ and $\mathbf{X}_t$ only differ in position I_t, each state transition can be generated by simply flipping the component of $\mathbf{X}_t$ at a randomly chosen index I_t. The following MATLAB program gives an implementation of the generation algorithm for a 20-dimensional hypercube. In Figure 4.3 the progress of the Markov chain $\mathbf{X}_t = (X_{t1}, \ldots, X_{tn})^\top$, $t = 0, 1, 2, \ldots, 200$ can be observed through its representation $Y_t = \sum_{i=1}^{n} 2^{-i} X_{ti}$, $t = 0, 1, 2, \ldots, 200$ on the interval $[0, 1]$.

```
%hypercube.m
N = 200; % number of samples
n = 20; %dimension
x = zeros(N,n);
for t=1:N
    I = ceil(rand*n); %choose random position
```

```
    x(t+1,:) = x(t,:); %copy
    x(t+1,I) = ~x(t+1,I); %flip bit at position I
end
b = 0.5.^[1:n];
y = x*b';
hold on
plot(0:N,y,'.'), plot(0:N,y)
hold off
```

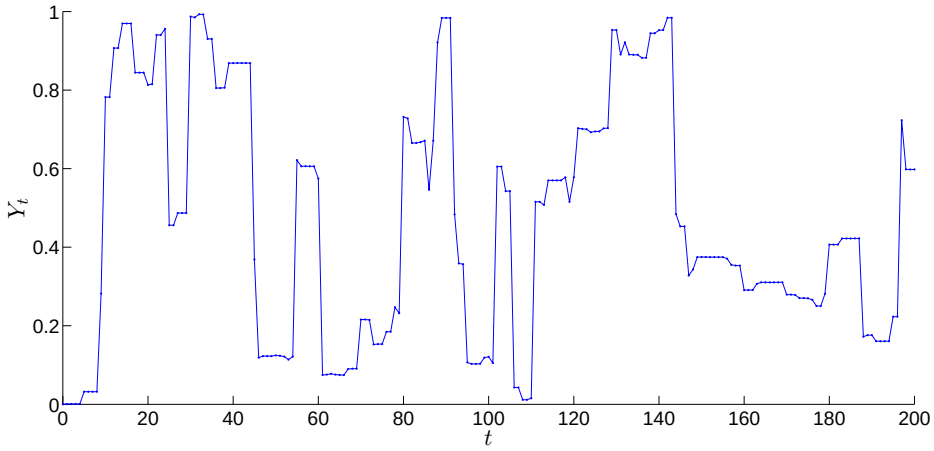


Figure 4.3: Realization of the process $\{Y_t\}$.

4.3 Markov Jump Processes

A **Markov jump process** is a stochastic process $\{X_t, t \in \mathscr{T}\}$ with a continuous index set $\mathscr{T} \subseteq \mathbb{R}$ and a discrete state space E, which satisfies the **Markov property**

$$(X_{t+s} \,|\, X_u, u \leqslant t) \sim (X_{t+s} \,|\, X_t) \,.$$

We discuss here only the main points pertinent to the simulation of such processes. We assume throughout that the index set is $\mathscr{T} = [0, \infty)$ and that the state space is $E = \{1, 2, \ldots\}$.

A time-homogeneous Markov jump process is often defined via its Q-matrix,

$$Q = \begin{pmatrix} -q_1 & q_{12} & q_{13} & \cdots \\ q_{21} & -q_2 & q_{23} & \cdots \\ q_{31} & q_{32} & -q_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} ,$$

where q_{ij} is the **transition rate** from i to j:

$$q_{ij} = \lim_{h \downarrow 0} \frac{\mathbb{P}(X_{t+h} = j \,|\, X_t = i)}{h} \,, \quad i \neq j, \quad i, j \in E \tag{4.2}$$

and q_i is the **holding rate** in i:

$$q_i = \lim_{h \downarrow 0} \frac{1 - \mathbb{P}(X_{t+h} = i \,|\, X_t = i)}{h} \,, \quad i \in E \,.$$

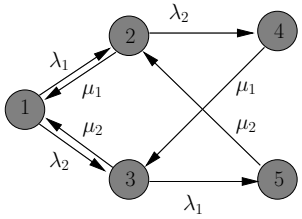
A typical assumption is that $0 \leqslant q_{ij} < \infty$ and that $q_i = \sum_{j \neq i} q_{ij}$, so that all row sums of Q are 0. The behavior of such a Markov jump process is as follows: if the process in some state i at time t, it will remain there for an additional $\mathsf{Exp}(q_i)$-distributed amount of time. When the process leaves a state i, it will jump to a state j with probability $K_{ij} = q_{ij}/q_i$, independent of the history of the process. In particular, the jump states $Y_0, Y_1, \ldots$ form a Markov chain with transition matrix $K = (K_{ij})$. Defining the holding times as $A_1, A_2, \ldots$ and the jump times as $T_1, T_2, \ldots$, the generation algorithm is as follows.

Algorithm 4.6 (Generating a Time-Homogeneous Markov Jump Process)

1. *Set $T_0 = 0$. Draw Y_0 from its distribution. Set $X_0 = Y_0$ and $n = 0$.*
2. *Draw $A_{n+1} \sim \mathsf{Exp}(q_{Y_n})$.*
3. *Set $T_{n+1} = T_n + A_{n+1}$.*
4. *Set $X_t = Y_n$ for $T_n \leqslant t < T_{n+1}$.*
5. *Draw Y_{n+1} from the distribution corresponding to the Y_n-th row of K. Set $n = n + 1$ and go to Step 2.*

Example 4.3 (Repairable System) Consider a reliability system with two unreliable machines and one repairman. Both machines have exponentially distributed life and repair times. The failure and repair rates are λ_1, μ_1 and λ_2, μ_2 for machine 1 and machine 2, respectively. The repairman can only work on one machine at a time, and if both have failed the repair man keeps working on the machine that has failed first, while the other machine remains idle. All life and repair times are independent of each other.

Because of the exponentiality and independence assumptions, the system can be described via a Markov jump process with 5 states: 1 (both machines working), 2 (machine 2 working, machine 1 failed), 3 (machine 1 working, machine 2 failed), 4 (both failed, machine 1 failed first), 5 (both failed, machine 2 failed first). The transition rate graph and Q-matrix are given in Figure 4.4.



$$Q = \begin{pmatrix} -(\lambda_1+\lambda_2) & \lambda_1 & \lambda_2 & 0 & 0 \\ \mu_1 & -(\mu_1+\lambda_2) & 0 & \lambda_2 & 0 \\ \mu_2 & 0 & -(\mu_2+\lambda_1) & 0 & \lambda_1 \\ 0 & 0 & \mu_1 & -\mu_1 & 0 \\ 0 & \mu_2 & 0 & 0 & -\mu_2 \end{pmatrix} .$$

Figure 4.4: Transition rate graph and Q-matrix of the repairable system.

The following MATLAB program implements Algorithm 4.6 for the case where $\lambda_1 = 1, \lambda_2 = 2$, $\mu_1 = 3$, and $\mu_4 = 4$. A realization of the process on the interval $[0, 5]$, starting in state 1, is given in Figure 4.5.

```
%mjprep.m
clear all, clf
lam1= 1; lam2 = 2; mu1= 3; mu2 = 4;
Q = [-(lam1 + lam2), lam1, lam2, 0, 0;
    mu1, -(mu1+ lam2), 0, lam2, 0;
    mu2, 0, -(mu2 + lam1), 0, lam1;
    0, 0, mu1, -mu1, 0;
    0, mu2, 0, 0, -mu2];

q = -diag(Q);
K = diag(1./q)*Q + eye(5);
T = 5;
n=0;
t = 0; y = 1;
yy = [y]; tt = [t];
while t < T
    A = -log(rand)/q(y);
    y =  min(find(cumsum(K(y,:))> rand));
    t = t + A;
    tt = [tt,t];
    yy= [yy,y];
    n= n+1;
end
for i=1:n
    line([tt(i),tt(i+1)],[yy(i),yy(i)],'Linewidth',3);
    line([tt(i+1),tt(i+1)],[yy(i),yy(i+1)],'LineStyle',':');
end
axis([0,T,1,5.1])
```

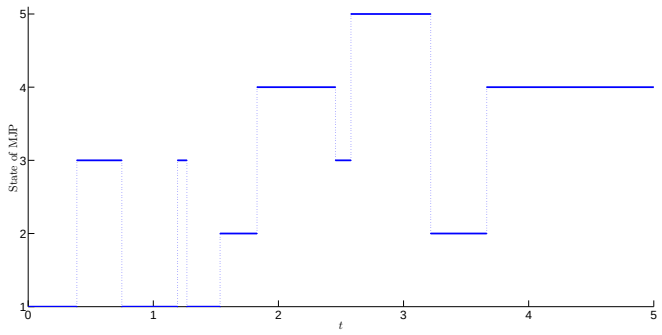


Figure 4.5: Realization of the reliability Markov jump process.

4.4 Poisson Processes

Poisson processes are used to model random configurations of points in space and time. Specifically, let E be some subset of $\mathbb{R}^d$ and let $\mathcal{E}$ be the collection of Borel sets on E. To any collection of random points $\{T_n\}$ in E corresponds a **random counting measure** N defined by

$$N(A) = \sum_k \mathrm{I}_{\{T_k \in A\}}, \quad A \in \mathcal{E} ,$$

counting the random number of points in A. Such a random counting measure is said to be a **Poisson random measure** with **mean measure** μ if the following properties hold:

1. $N(A) \sim \mathsf{Poi}(\mu(A))$ for any set $A \in \mathcal{E}$, where $\mu(A)$ denotes the mean measure of A.

2. For any disjoint sets $A_1, \ldots, A_n \in \mathcal{E}$, the random variables $N(A_1), \ldots, N(A_n)$ are independent.

In most practical cases the mean measure has a density, called the **intensity** or **rate** function, $\lambda(\mathbf{x})$; so that

$$\mu(A) = \int_A \lambda(\mathbf{x}) \, \mathrm{d}\mathbf{x} .$$

We will assume from now on that such a rate function exists.

Informally, both the collection $\{T_k\}$ and the random measure N are referred to as a **Poisson process** on E. The Poisson process is said to be **homogeneous** if the rate function is constant. An important corollary of Properties 1 and 2 above is:

3. Conditional upon $N(A) = n$, the n points in A are independent of each other and have pdf $f(\mathbf{x}) = \lambda(\mathbf{x})/\mu(A)$.

This leads immediately to the following generic algorithm for generating a Poisson process on E, assuming that $\mu(E) = \int_E \lambda(\mathbf{x}) \, \mathrm{d}\mathbf{x} < \infty$.

Algorithm 4.7 (Generating a General Poisson Random Measure)

1. *Generate a Poisson random variable* $N \sim \mathsf{Poi}(\mu(E))$.

2. *Given* $N = n$, *draw* $\mathbf{X}_1, \ldots, \mathbf{X}_n \overset{\text{iid}}{\sim} f$, *where* $f(\mathbf{x})$ *is the mean density* $\lambda(\mathbf{x})/\mu(E)$, *and return these as the points of the Poisson process.*

Example 4.4 (Convex Hull of a Poisson Process) Figure 4.6 shows six realizations of the point sets and their *convex hulls* of a homogeneous Poisson process on the unit square with rate 20. The MATLAB code is given below. A particular object of interest could be the random volume of the convex hull formed in this way.

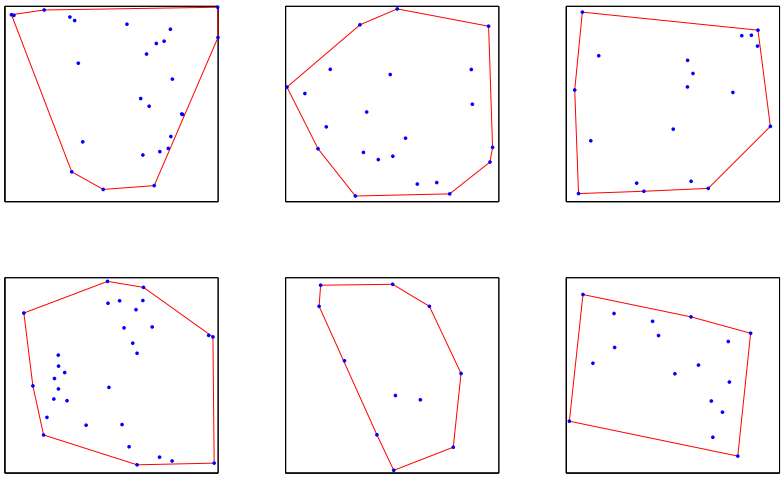


Figure 4.6: Realizations of a homogeneous Poisson process with rate 20. For each case the convex hull is also plotted.

```
%hompoich.m
for i=1:6
   N = poissrnd(20);
   x = rand(N,2);
   k = convhull(x(:,1),x(:,2));
   %[K,v] =  convhulln(x); %v is the area
   subplot(2,3,i);
   plot(x(k,1),x(k,2),'r-',x(:,1),x(:,2),'.')
end
```

For one-dimensional Poisson processes more direct generation algorithms can be formulated, using the additional properties of such processes. Consider first a homogeneous Poisson process with rate λ on $\mathbb{R}_+$. Denote the points of the process by $0 < T_1, T_2, \ldots$, interpreted as *arrival* points of some sort, and let $A_i = T_i - T_{i-1}$ be the i-th *interarrival* time, $i = 1, 2, \ldots$, setting $T_0 = 0$. The interarrival times $\{A_i\}$ are iid and $\mathsf{Exp}(\lambda)$ distributed. We can thus generate the points of the Poisson process on some interval $[0, T]$ as follows.

Algorithm 4.8 (One-Dimensional Homogeneous Poisson Process)

1. *Set* $T_0 = 0$ *and* $n = 1$.
2. *Generate* $U \sim \mathsf{U}(0, 1)$.
3. *Set* $T_n = T_{n-1} - \frac{1}{\lambda} \ln U$.
4. *If* $T_n > T$, *stop; otherwise, set* $n = n + 1$ *and go to Step 2.*

Notice that the corresponding **Poisson counting process** $\{N_t, t \geqslant 0\}$, defined by $N_t = N([0,t])$, is a Markov jump process on $\{0,1,2,\ldots\}$ with $N_0 = 0$ and transition rates $q_{i,i+1} = \lambda$, $i = 0,1,2,\ldots$ and $q_{i,j} = 0$ otherwise. The process jumps at times $T_1, T_2, \ldots$ to states $1,2,\ldots$, staying an $\mathsf{Exp}(\lambda)$-distributed amount of time in each state (including in state 0). In a similar way, the counting process corresponding to a *non-homogeneous* one-dimensional Poisson process on $\mathbb{R}_+$ with rate function $\lambda(t), t \geqslant 0$ is a non-homogeneous Markov jump process with transition rates $q_{i,i+1}(t) = \lambda(t)$, $i = 0,1,2,\ldots$. The tail probabilities of the interarrival times are now

$$\mathbb{P}(A_{n+1} > t) = \exp\left(-\int_{T_n}^{T_n+t} \lambda(s)\,\mathrm{d}s \right), \quad t \geqslant 0\,.$$

Therefore, a variety of generation methods are available to generate the interarrival times directly. However, it is often easier to construct the points indirectly, as illustrated in Figure 4.7: First select a constant $\lambda \geqslant \sup_{s \leqslant t} \lambda(s)$, assuming it exists. Then, generate the points of a two-dimensional homogeneous Poisson process, M say, on $[0,t] \times [0,\lambda]$ with rate 1. Finally, project all points of M that lie below the graph of $\lambda(s)$, $s \leqslant t$ onto the t-axis.

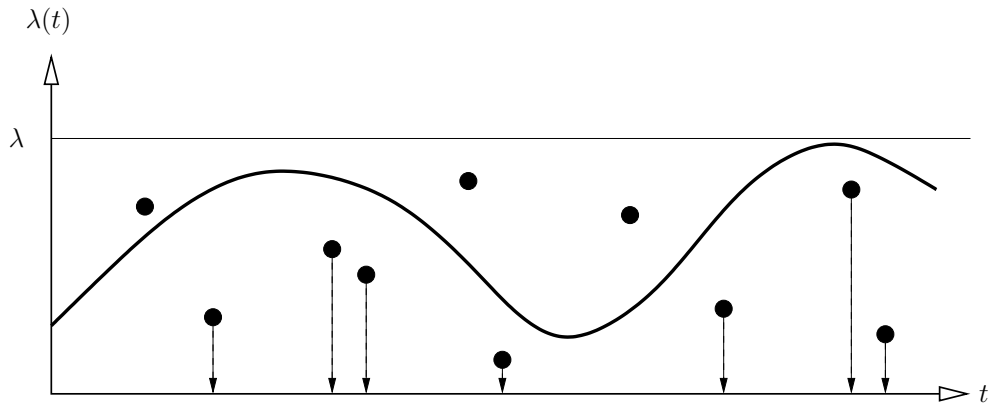


Figure 4.7: Constructing a non-homogeneous Poisson process.

Let $\mathscr{R}_t = \{(s,y), 0 \leqslant s \leqslant t, y \leqslant \lambda(s)\}$. For each $t \geqslant 0$, we have

$$\mathbb{P}(N_t = 0) = \mathbb{P}(M(\mathscr{R}_t) = 0) = \exp\left(-\int_0^t \lambda(s)\,\mathrm{d}s \right),$$

which shows that the stochastic process $\{N_t, t \geqslant 0\}$ constructed in this way is a non-homogeneous Poisson counting process with rate function $\lambda(t), t \geqslant 0$. If instead *all* points of M are projected onto the t-axis, we obtain a homogeneous Poisson counting process with rate λ. To obtain the non-homogeneous process we accept each point τ with probability $\frac{\lambda(\tau)}{\lambda}$. This leads to the following algorithm.

Algorithm 4.9 (One-Dimensional Non-Homogeneous Poisson Process)

1. *Set* $t = 0$ *and* $n = 0$.
2. *Generate* $U \sim \mathsf{U}(0,1)$.
3. *Set* $t = t - \frac{1}{\lambda} \ln U$.
4. *If* $t > T$, *stop; otherwise, continue.*
5. *Generate* $V \sim \mathsf{U}(0,1)$.
6. *If* $V \leqslant \frac{\lambda(t)}{\lambda}$, *increase* n *by 1 and set* $T_n = t$. *Repeat from Step 2.*

Example 4.5 (A Non-Homogeneous Poisson Counting Process) Figure 4.8 gives a typical realization of a non-homogeneous Poisson counting process $\{N_t, t \geqslant 0\}$ with rate function $\lambda(t) = \sin^2(t)$ on the interval $[0, 50]$. The realization is obtained using the following MATLAB code, which implements Algorithm 4.9.

```
%pois.m
T = 50; t = 0; n = 0;
tt = [t];
while t < T
    t = t - log(rand);
      if (rand < sin(t)^2)
        tt = [tt,t];
        n = n+1;
    end
end
nn = 0:n;
for i =1:n
    line([tt(i),tt(i+1)],[nn(i),nn(i)],'Linewidth',2);
end
```

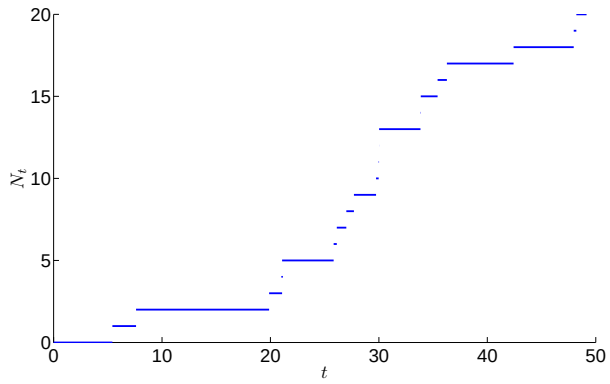


Figure 4.8: A typical realization of a non-homogeneous Poisson counting process with rate function $\lambda(t) = \sin^2(t)$.

4.5 Wiener Process and Brownian Motion

A **Wiener process** is a stochastic process $W = \{W_t, t \geqslant 0\}$ characterized by the following properties.

1. *Independent increments*: W has **independent increments**; that is, for any $t_1 < t_2 \leqslant t_3 < t_4$

$$W_{t_4} - W_{t_3} \quad \text{and} \quad W_{t_2} - W_{t_1}$$

   are independent random variables. In other words, $W_t - W_s,\ t > s$ is independent of the past history of $\{W_u,\ 0 \leqslant u \leqslant s\}$.

2. *Gaussian stationarity*: For all $t \geqslant s \geqslant 0$,

$$W_t - W_s \sim \mathsf{N}(0, t-s)\ .$$

3. *Continuity of paths*: $\{W_t\}$ has continuous paths, with $W_0 = 0$.

The Wiener process plays a central role in probability and forms the basis of many other stochastic processes. It can be viewed as a continuous version of a random walk process. Two typical sample paths are depicted in Figure 4.9. ☞ 65

Remark 4.5.1 (Starting Position) Although by definition the Wiener process starts at position 0, it is useful to consider Wiener processes starting from some arbitrary state x under a probability measure $\mathbb{P}^x$.

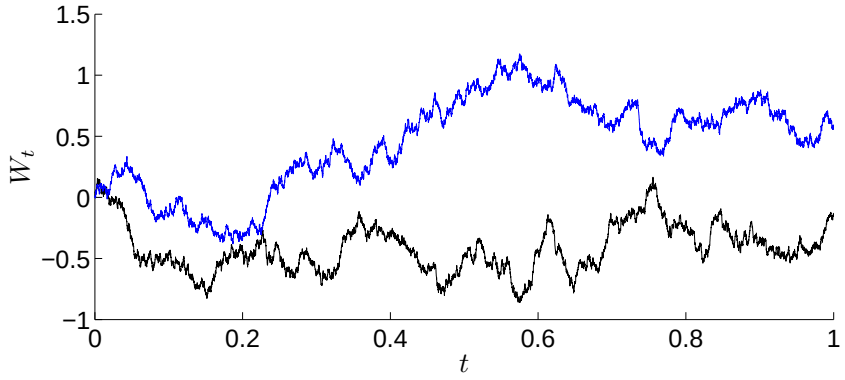


Figure 4.9: Two realizations of the Wiener process on the interval [0,1].

The two main properties of the Wiener process $W = \{W_t, t \geqslant 0\}$ are

1. *Gaussian process*: W is a Gaussian process with $\mathbb{E}W_t = 0$ and $\mathrm{Cov}(W_s, W_t) = \min\{s,t\}$. It is the only Gaussian process with continuous sample paths that has these properties.

2. *Markov property*: W is a time-homogeneous strong Markov process. In particular, for any finite stopping time τ and for all $t \geqslant 0$,

$$(W_{\tau+t} \,|\, W_u, u \leqslant \tau) \sim (W_{\tau+t} \,|\, W_\tau)\ .$$

The basic generation algorithm below uses the Markovian and Gaussian properties of the Wiener process.

Algorithm 4.10 (Generating the Wiener Process)

1. *Let* $0 = t_0 < t_1 < t_2 < \cdots < t_n$ *be the set of distinct times for which simulation of the process is desired.*
2. *Generate* $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$ *and output*

$$W_{t_k} = \sum_{i=1}^{k} \sqrt{t_k - t_{k-1}}\, Z_i, \quad k = 1, \ldots, n\,.$$

The algorithm is *exact*, in that the $\{W_{t_k}\}$ are drawn exactly according to their respective distributions. Nevertheless, the algorithm returns only a discrete skeleton of the true continuous process. To obtain a continuous path approximation to the exact path of the Wiener process, one could use linear interpolation on the points $W_{t_1}, \ldots, W_{t_n}$. In other words, within each interval $[t_{k-1}, t_k],\ k = 1, \ldots, n$ approximate the continuous process $\{W_s, s \in [t_{k-1}, t_k]\}$ via:

$$\widehat{W}_s = \frac{W_{t_k}(s - t_{k-1}) + W_{t_{k-1}}(t_k - s)}{(t_k - t_{k-1})}, \quad s \in [t_{k-1}, t_k]\,.$$

It is possible to adaptively refine the path by using a Brownian bridge process, see Section 4.7.

A process $\{B_t, t \geqslant 0\}$ satisfying

$$B_t = \mu\, t + \sigma\, W_t, \quad t \geqslant 0\,,$$

where $\{W_t\}$ is a Wiener process, is called a **Brownian motion** with **drift** μ and **diffusion coefficient** σ^2. It is called a **standard Brownian motion** if $\mu = 0$ and $\sigma^2 = 1$ (Wiener process).

The generation of a Brownian motion at times $t_1, \ldots, t_n$ follows directly from its definition.

Algorithm 4.11 (Generating Brownian Motion)

1. *Generate outcomes* $W_{t_1}, \ldots, W_{t_n}$ *of a Wiener process at times* $t_1, \ldots, t_n$.
2. *Return* $B_{t_i} = \mu\, t_i + \sigma\, W_{t_i}$, $i = 1, \ldots, n$ *as the outcomes of the Brownian motion at times* $t_1, \ldots, t_n$.

Let $\{W_{t,i}, t \geqslant 0\}$, $i = 1, \ldots, n$ be independent Wiener processes and let $\mathbf{W}_t = (W_{t,1}, \ldots, W_{t,n})$. The process $\{\mathbf{W}_t, t \geqslant 0\}$ is called an ***n*-dimensional Wiener process**.

Example 4.6 (Three-Dimensional Wiener Process) The following MATLAB program generates a realization of the three-dimensional Wiener process at times $0, 1/N, 2/N, \ldots, 1$, for $N = 10^4$. Figure 4.10 shows a typical realization.

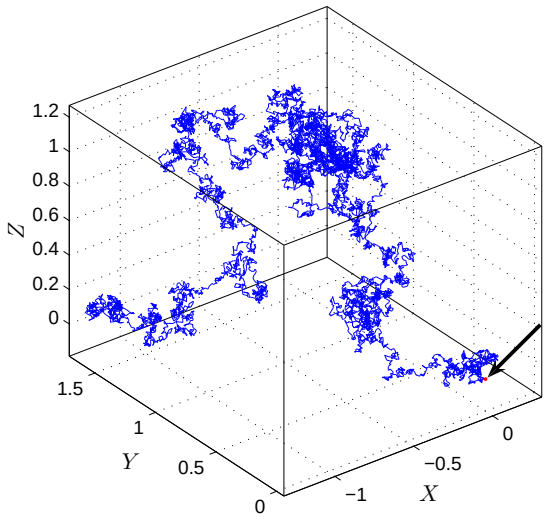


Figure 4.10: Three-dimensional Wiener process $\{\mathbf{W}_t, 0 \leqslant t \leqslant 1\}$. The arrow points at the origin.

```
%wp3d.m
N=10^4; T=1; dt=T/N; %step size
X=cumsum([0,0,0;randn(N,3)*sqrt(dt)],1);
plot3(X(:,1),X(:,2),X(:,3))
```

4.6 Stochastic Differential Equations and Diffusion Processes

A **stochastic differential equation** (SDE) for a stochastic process $\{X_t, t \geqslant 0\}$ is an expression of the form

$$\mathrm{d}X_t = a(X_t, t)\,\mathrm{d}t + b(X_t, t)\,\mathrm{d}W_t\ , \tag{4.3}$$

where $\{W_t, t \geqslant 0\}$ is a Wiener process and $a(x,t)$ and $b(x,t)$ are deterministic functions. The coefficient (function) a is called the **drift** and b^2 is called the **diffusion** coefficient — some authors call b the diffusion coefficient. The resulting process $\{X_t, t \geqslant 0\}$ — which is a Markov process with continuous sample paths — is referred to as an **(Itô) diffusion**.

Stochastic differential equations are based on the same principle as ordinary differential equations, relating an unknown function to its derivatives, but with the additional feature that part of the unknown function is driven by randomness. Intuitively, (4.3) expresses that the infinitesimal change in $\mathrm{d}X_t$ at time t is the sum of an infinitesimal displacement $a(X_t, t)\,\mathrm{d}t$ and an infinitesimal noise term $b(X_t, t)\,\mathrm{d}W_t$. The precise mathematical meaning of (4.3) is that the

stochastic process $\{X_t, t \geqslant 0\}$ satisfies the integral equation

$$X_t = X_0 + \int_0^t a(X_s, s)\,\mathrm{d}s + \int_0^t b(X_s, s)\,\mathrm{d}W_s\ , \tag{4.4}$$

where the last integral is an **Itô integral**.

Multidimensional SDEs can be defined in a similar way as in (4.3). A stochastic differential equation in $\mathbb{R}^m$ is an expression of the form

$$\mathrm{d}\mathbf{X}_t = \mathbf{a}(\mathbf{X}_t, t)\,\mathrm{d}t + B(\mathbf{X}_t, t)\,\mathrm{d}\mathbf{W}_t\ , \tag{4.5}$$

where $\{\mathbf{W}_t\}$ is an n-dimensional Wiener process, $\mathbf{a}(\mathbf{x}, t)$ is an m-dimensional vector (the drift) and $B(\mathbf{x}, t)$ an $m \times n$ matrix, for each $\mathbf{x} \in \mathbb{R}^m$ and $t \geqslant 0$. The $m \times m$ matrix $C = BB^\top$ is called the **diffusion matrix**. The resulting diffusion process is Markov, and if $\mathbf{a}$ and B do not depend explicitly on t then the diffusion process is time-homogeneous.

We next discuss a simple technique for approximately simulating diffusion processes.

4.6.1 Euler's Method

Let $\{X_t, t \geqslant 0\}$ be a diffusion process defined by the SDE

$$\mathrm{d}X_t = a(X_t, t)\,\mathrm{d}t + b(X_t, t)\,\mathrm{d}W_t\ , \quad t \geqslant 0\ , \tag{4.6}$$

where X_0 has a known distribution.

The **Euler** or **Euler–Maruyama** method for solving SDEs is a simple generalization of Euler's method for solving ordinary differential equations. The idea is to replace the SDE with the stochastic difference equation

$$Y_{k+1} = Y_k + a(Y_k, kh)\,h + b(Y_k, kh)\sqrt{h}\,Z_k\ , \tag{4.7}$$

where $Z_1, Z_2, \ldots \sim_{\text{iid}} \mathsf{N}(0,1)$. For a small step size h the time series $\{Y_k, k = 0, 1, 2, \ldots\}$ approximates the process $\{X_t, t \geqslant 0\}$; that is, $Y_k \approx X_{kh}$, $k = 0, 1, 2, \ldots$.

Algorithm 4.12 (Euler's Method)

1. *Generate Y_0 from the distribution of X_0. Set $k = 0$.*
2. *Draw $Z_k \sim \mathsf{N}(0,1)$.*
3. *Evaluate Y_{k+1} from (4.7) as an approximation to X_{kh}.*
4. *Set $k = k + 1$ and go to Step 2.*

Remark 4.6.1 (Interpolation) The Euler procedure only returns approximations to $\{X_t\}$ at times that are multiples of the step size h. To obtain approximations for times $s \neq kh$ one could simply approximate X_t by Y_k for $t \in [kh, (k+1)h)$, or use the linear interpolation

$$\left(k + 1 - \frac{t}{h}\right) Y_k + \left(\frac{t}{h} - k\right) Y_{k+1}, \quad t \in [kh, (k+1)h]\ .$$

For a multidimensional SDE of the form (4.5) the Euler method has the following simple generalization.

Algorithm 4.13 (Multidimensional Euler Method)

1. *Generate* $\mathbf{Y}_0$ *from the distribution of* $\mathbf{X}_0$. *Set* $k = 0$.
2. *Draw* $\mathbf{Z}_k \sim \mathsf{N}(\mathbf{0}, I)$.
3. *Evaluate*
$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mathbf{a}(\mathbf{Y}_k, kh)\, h + B(\mathbf{Y}_k, kh)\, \sqrt{h}\, \mathbf{Z}_k$$
as an approximation to $\mathbf{X}_{kh}$.
4. *Set* $k = k + 1$ *and go to Step 2.*

Example 4.7 (Simplified Duffing–Van der Pol Oscillator) Consider the following two-dimensional SDE:

$$\begin{aligned} \mathrm{d}X_t &= Y_t\, \mathrm{d}t\,, \\ \mathrm{d}Y_t &= \left(X_t \left(\alpha - X_t^2\right) - Y_t\right)\, \mathrm{d}t + \sigma X_t\, \mathrm{d}W_t\,. \end{aligned}$$

The following MATLAB code generates the process with parameters $\alpha = 1$ and $\sigma = 1/2$ for $t \in [0, 1000]$, using a step size $h = 10^{-3}$ and starting at $(-2, 0)$. Note that the process oscillates between two modes.

```
%vdpol.m
alpha = 1; sigma = 0.5;
a1 = @(x1,x2,t) x2;
a2 = @(x1,x2,t) x1*(alpha-x1^2)-x2;
b1 = @(x1,x2,t) 0 ;
b2 = @(x1,x2,t) sigma*x1;
n=10^6; h=10^(-3); t=h.*(0:1:n); x1=zeros(1,n+1); x2=x1;
x1(1)=-2;
x2(1)=0;
for k=1:n
   x1(k+1)=x1(k)+a1(x1(k),x2(k),t(k))*h+ ...
           b1(x1(k),x2(k),t(k))*sqrt(h)*randn;
   x2(k+1)=x2(k)+a2(x1(k),x2(k),t(k))*h+ ...
           b2(x1(k),x2(k),t(k))*sqrt(h)*randn;
end
step = 100; %plot each 100th value
figure(1),plot(t(1:step:n),x1(1:step:n),'k-')
figure(2), plot(x1(1:step:n),x2(1:step:n),'k-');
```

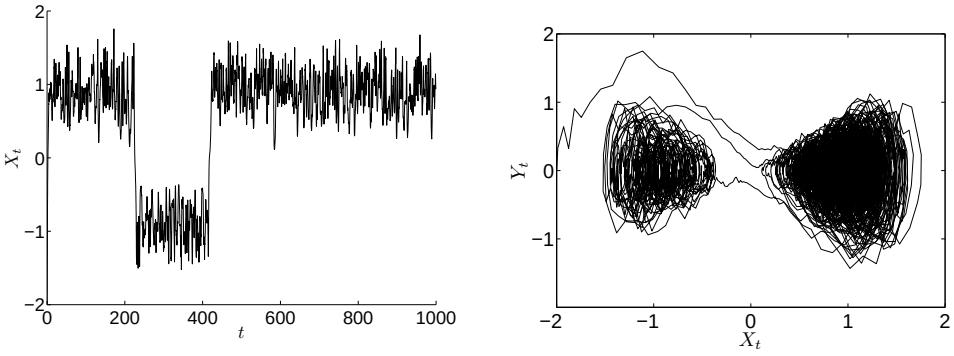


Figure 4.11: A typical realization with parameters $\alpha = 1$ and $\sigma = 1/2$, starting at $(-2, 0)$.

4.7 Brownian Bridge

The **standard Brownian bridge** process $\{X_t, t \in [0,1]\}$ is a stochastic process whose distribution is that of the Wiener process on $[0,1]$ *conditioned* upon $X_1 = 0$. In other words, the Wiener process is "tied-down" to be 0 at both time 0 and time 1. The standard Brownian bridge process is a Gaussian process with mean 0 and covariance function $\mathrm{Cov}(X_s, X_t) = \min\{s,t\} - s\,t$.

It is not difficult to show that if $\{W_t\}$ is a Wiener process, then,

$$X_t = W_t - t\,W_1, \quad 0 \leqslant t \leqslant 1\,,$$

defines a standard Brownian bridge.

A realization of the process is given in Figure 4.12.

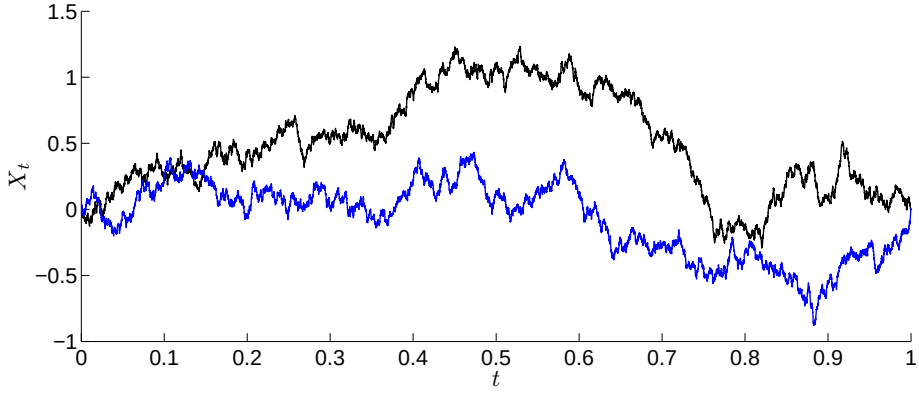


Figure 4.12: Two realizations of the standard Brownian bridge process on the interval $[0,1]$.

Generation of a sample path of the standard Brownian bridge is most easily accomplished by the following algorithm.

Algorithm 4.14 (Generating a Standard Brownian Bridge)

1. *Let* $0 = t_0 < t_1 < t_2 < \cdots < t_{n+1} = 1$ *be the set of distinct times for which simulation of the process is desired.*
2. *Generate a Wiener process on* $t_1, \ldots, t_n$ *using, for example, Algorithm 4.10.*
3. *Set* $X_0 = 0$ *and* $X_1 = 0$*, and output for each* $k = 1, \ldots, n$*:*

$$X_{t_k} = W_{t_k} - t_k \, W_{t_n}.$$

A general (non-standard) **Brownian bridge** is a stochastic process $\{X_t, t \in [t_0, t_{n+1}]\}$ whose distribution in that of the Wiener process on $[t_0, t_{n+1}]$ conditioned upon $X_{t_0} = a$ and $X_{t_{n+1}} = b$. The general Brownian bridge is a Gaussian process with mean function

$$\mu_t = a + \frac{(b-a)(t-t_0)}{(t_{n+1} - t_0)}$$

and covariance function

$$\mathrm{Cov}(X_s, X_t) = \min\{s - t_0,\, t - t_0\} - \frac{(s-t_0)(t-t_0)}{(t_{n+1} - t_0)} \,.$$

A frequent use of the Brownian bridge is for adaptive refinement of the discrete-time approximation of the Wiener process. Suppose that we have generated the Wiener process at certain time instants and wish to generate the process for additional times. Specifically, suppose $W_{t_0} = a$ and $W_{t_{n+1}} = b$ and we wish to generate the Wiener process for additional times $t_1, \ldots, t_n$ in the interval $[t_0, t_{n+1}]$.

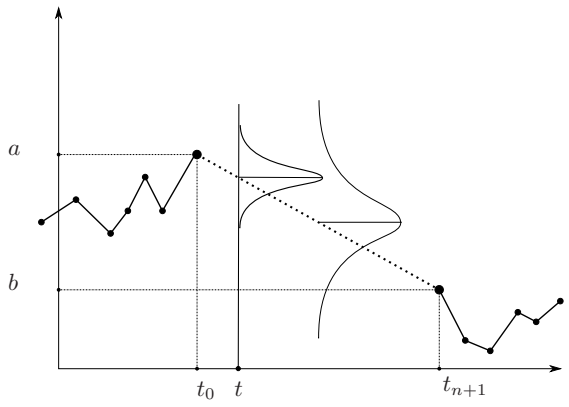


Figure 4.13: Conditional on $W_{t_0} = a$ and $W_{t_{n+1}} = b$, the point $W_t, t \in [t_0, t_{n+1}]$ has a normal distribution.

Conditional on $W_{t_0} = a$ and $W_{t_{n+1}} = b$, the process $\{W_t, t \in [t_0, t_{n+1}]\}$ is a Brownian bridge; see Figure 4.13. In particular, the distribution of W_t with $t \in [t_0, t_{n+1}]$ is normal with mean $a + (b-a)(t-t_0)/(t_{n+1} - t_0)$ and variance

$(t_{n+1}-t)(t-t_0)/(t_{n+1}-t_0)$. We can thus generate the process at any number of additional points $t_1 < \cdots < t_n$ within the interval $[t_0, t_{n+1}]$ using the following algorithm.

Algorithm 4.15 (Brownian Bridge Sampling Between W_{t_0} and $W_{t_{n+1}}$)

1. *Generate* $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$.
2. *For each* $k = 1, \ldots, n$, *output:*

$$W_{t_k} = W_{t_{k-1}} + (b - W_{t_{k-1}}) \frac{t_k - t_{k-1}}{t_{n+1} - t_{k-1}} + \sqrt{\frac{(t_{n+1} - t_k)(t_k - t_{k-1})}{t_{n+1} - t_{k-1}}}\, Z_k \,.$$

In principle we need not generate the intermediate points $W_{t_1}, \ldots, W_{t_n}$ in any particular order as long as at each step we condition on the two closest time points already sampled. The following code implements the algorithm given above.

```
function X=brownian_bridge(t,x_r,x_s,Z)
n=length(t)-2;X=nan(1,n+2);
X(1)=x_r; X(n+2)=x_s;s=t(n+2);
for k=2:n+1
    mu=X(k-1)+(x_s-X(k-1))*(t(k)-t(k-1))/(s-t(k-1));
    sig2=(s-t(k))*(t(k)-t(k-1))/(s-t(k-1));
    X(k)=mu+sqrt(sig2)*Z(k-1);
end
```

4.8 Geometric Brownian Motion

The **geometric Brownian motion** process satisfies the homogeneous linear SDE

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

which has strong solution

$$X_t = X_0\, \mathrm{e}^{(\mu - \frac{\sigma^2}{2})\, t + \sigma W_t}, \quad t \geqslant 0 \,.$$

The special case where $\mu = 0$ and $\sigma = 1$ is called the **stochastic exponential** of the Wiener process $\{W_t\}$.

An important application is the **Black–Scholes** equity model, where X_t is the price of the equity at time t, and the interest rate at time t is modeled as the sum of a fixed rate μ and an uncertain rate $\sigma\eta_t$, with $\eta_t = \mathrm{d}W_t/\mathrm{d}t$ denoting Gaussian "white noise" and σ a volatility factor.

One usually takes $X_0 = 1$. The expectation and variance of X_t are then given by

$$\mathbb{E}X_t = \mathrm{e}^{\mu t} \qquad \text{and} \qquad \mathrm{Var}(X_t) = \mathrm{e}^{2\mu t}\left(\mathrm{e}^{\sigma^2 t} - 1\right) \,.$$

The strong solution of the geometric Brownian motion suggests the following exact simulation algorithm.

Algorithm 4.16 (Geometric Brownian Motion) *Let* $0 = t_0 < t_1 < t_2 < \cdots < t_n$ *be the set of distinct times for which simulation of the process is desired.*

1. *Generate* $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1)$.

2. *Output*

$$X_{t_k} = X_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) t_k + \sigma \sum_{i=1}^{k} \sqrt{t_i - t_{i-1}}\, Z_i\right), \quad k = 1, \ldots, n\,.$$

The following code implements the algorithm.

```
%geometricbm.m
T=1; % final time
n=10000; h=T/(n-1); t= 0:h:T;
mu = 1; sigma = 0.2; xo=1; % parameters
W = sqrt(h)*[0, cumsum(randn(1,n-1))];
x = xo*exp((mu - sigma^2/2)*t + sigma*W);
plot(t,x)
hold on
plot(t, exp(mu*t),'r'); %plot exact mean function
```

Figure 4.14 depicts two realizations of a geometric Brownian motion on the interval $[0, 1]$, with parameters $\mu = 1$ and $\sigma = 0.2$.

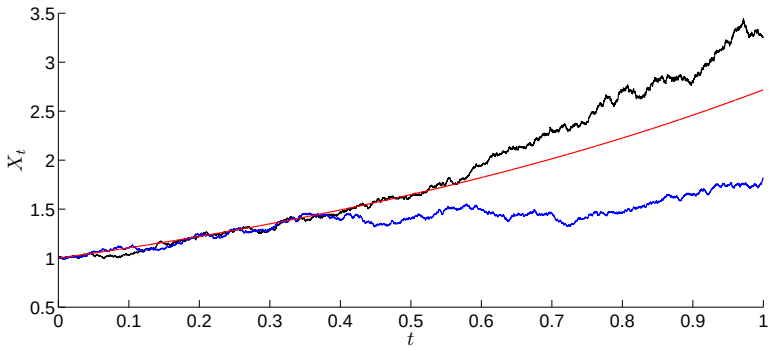


Figure 4.14: Two realizations of a geometric Brownian motion on the interval $[0, 1]$, with parameters $\mu = 1$ and $\sigma = 0.2$. The smooth line depicts the expectation function.

4.9 Ornstein–Uhlenbeck Process

The **Ornstein–Uhlenbeck** process satisfies the SDE

$$\mathrm{d}X_t = \theta(\nu - X_t)\,\mathrm{d}t + \sigma\,\mathrm{d}W_t\ , \tag{4.8}$$

with $\sigma > 0$, $\theta > 0$, and $\nu \in \mathbb{R}$. The (strong) solution of this linear SDE is given by

$$X_t = \mathrm{e}^{-\theta t}X_0 + \nu(1 - \mathrm{e}^{-\theta t}) + \sigma\mathrm{e}^{-\theta t}\int_0^t \mathrm{e}^{\theta s}\,\mathrm{d}W_s\ .$$

It follows that $\{X_t\}$ is a Gaussian process whenever X_0 is Gaussian, with mean function

$$\mathbb{E}X_t = \mathrm{e}^{-\theta t}\,\mathbb{E}X_0 + \nu(1 - \mathrm{e}^{-\theta t})$$

and covariance function

$$\mathrm{Cov}(X_s, X_t) = \frac{\sigma^2}{2\theta}\,\mathrm{e}^{-\theta(s+t)}\left(\mathrm{e}^{2\theta\min\{s,t\}} - 1\right)\ .$$

In particular,

$$\mathrm{Var}(X_t) = \sigma^2\frac{1 - \mathrm{e}^{-2\theta t}}{2\theta}\ .$$

This shows that X_t converges in distribution to a $\mathsf{N}(\nu, \sigma^2/(2\theta))$ random variable as $t \to \infty$. It can be shown that

$$X_t = \mathrm{e}^{-\theta t}X_0 + \nu(1 - \mathrm{e}^{-\theta t}) + \sigma\mathrm{e}^{-\theta t}\,W\left(\frac{\mathrm{e}^{2t\theta} - 1}{2\theta}\right), \quad t \geqslant 0$$

defines an Ornstein–Uhlenbeck process, where $\{W(t) = W_t\}$ is a Wiener process. In particular, if $Y_t = X_t - \nu(1 - \mathrm{e}^{-\theta t})$, $t \geqslant 0$, then $\{Y_t\}$ is an Ornstein–Uhlenbeck process with $\nu = 0$, $Y_0 = X_0$ and exact solution:

$$Y_t = \mathrm{e}^{-\theta t}Y_0 + \sigma W\left(\frac{1 - \mathrm{e}^{-2t\theta}}{2\theta}\right), \quad t \geqslant 0\ . \tag{4.9}$$

It follows that if we can simulate from an Ornstein–Uhlenbeck process with $\nu = 0$, say $\{Y_t\}$, then we can also simulate from an Ornstein–Uhlenbeck process, say $\{X_t\}$, with $\nu \neq 0$. The following exact algorithm simulates $\{Y_t\}$ exactly and then uses the relation $X_t = Y_t + \nu(1 - \mathrm{e}^{-\theta t})$ to construct a sample path for $\{X_t\}$. The algorithm simulates $\{X_t\}$ exactly for any discretization.

Algorithm 4.17 (Generating an Ornstein–Uhlenbeck Process) *Let* $0 = t_0 < t_1 < t_2 < \cdots < t_n$ *be the set of distinct times for which simulation of the process is desired.*

1. *If* X_0 *is random, draw* X_0 *from its distribution. Set* $Y_0 = X_0$.
2. *For* $k = 1, 2, \ldots, n$ *compute:*

$$Y_k = \mathrm{e}^{-\theta(t_k - t_{k-1})}Y_{k-1} + \sigma\sqrt{\frac{1 - \mathrm{e}^{-2\theta(t_k - t_{k-1})}}{2\theta}}\ Z_k\ ,$$

where $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0, 1)$.

3. *Output* $(X_0, X_{t_1}, \ldots, X_{t_n})$, *where*

$$X_{t_k} = Y_k + \nu(1 - \mathrm{e}^{-\theta t_k}), \quad k = 1, \ldots, n\ .$$

A realization of the process is given in Figure 4.15 using three different starting conditions. The following code implements the algorithm.

```
%ou_timechange_ex.m
T=4; % final time
N=10^4; % number of steps
theta=2;nu=1;sig=0.2; % parameters
x=nan(N,1); x(1)=0;     % initial point
h=T/(N-1); % step size
t=0:h:T;   % time
% code the right-hand side of the updating formula
f=@(z,x)(exp(-theta*h)*x+sig*...
          sqrt((1-exp(-2*h*theta))/(2*theta))*z);
for i=2:N
     x(i)=f(randn,x(i-1));
end
x=x+nu*(1-exp(-theta*t'));
plot(t,x)
```

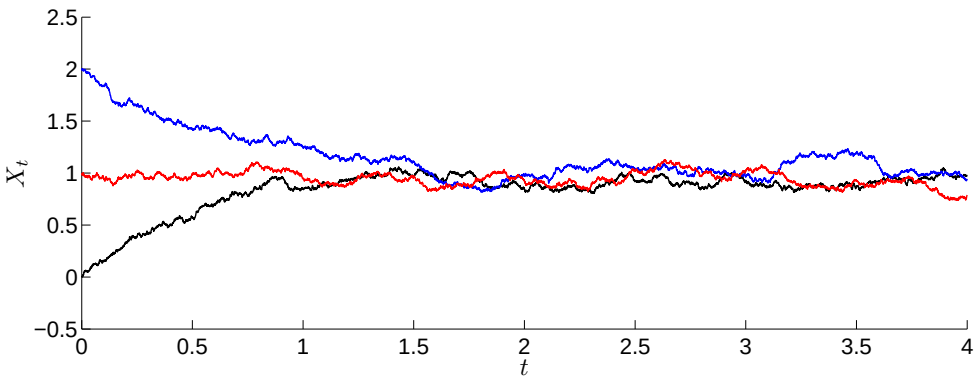


Figure 4.15: Three realizations of an Ornstein–Uhlenbeck process with parameters $\nu = 1, \theta = 2$, and $\sigma = 0.2$, starting from $0, 1$, and 2. After about 2 time units all paths have reached "stationarity" and fluctuate around the long-term mean ν.

The Ornstein–Uhlenbeck process has applications in physics and finance. For example, X_t is used to describe the *velocity* of a Brownian particle. The term ν is then usually taken to be 0, and the resulting SDE is said to be of *Langevin* type. In finance, the process is used to describe price fluctuations around a mean price ν such that the process "reverts to the mean" — that is, when $X_t > \nu$ the drift is negative, and when $X_t < \nu$ the drift is positive, so that at all times the process tends to drift toward ν.

4.10 Exercises

1. Write a program to generate realizations of a random walk where, starting from position 0, at each time instant a step of size 1 is taken to the left or right of the current position with equal probability, independently of the history of the process.

2. Consider a random walk on the undirected graph in Figure 4.16. For example, if the random walk at some time is in state 5, it will jump to 3, 4, or 6 at the next transition, each with probability 1/3.

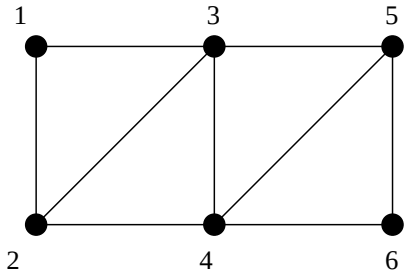


Figure 4.16: A graph.

(a) Simulate the random walk on a computer and verify that, in the long run, the proportion of visits to the various nodes is in accordance with the stationary distribution $\pi = (\frac{1}{9}, \frac{1}{6}, \frac{2}{9}, \frac{2}{9}, \frac{1}{6}, \frac{1}{9})$.

(b) Generate various sample paths for the random walk on the integers for $p = 1/2$ and $p = 2/3$.

3. A *birth and death process* is a Markov jump process whose Q matrix is a tri-diagonal. Thus from each state i only transitions $i \to i+1$ and $i \to i-1$ are allowed, with rates b_i (a birth) and d_i (a death), respectively. Write a MATLAB program to draw realizations from the following birth and death processes:

(a) The M/M/1 queue. Here $b_i = \lambda > 0$, $i = 0, 1, 2, \ldots$ and $d_i = \mu > 0$, $i = 1, 2, \ldots$. Try different values for λ and μ.

(b) The Yule process. Here $b_i = (n-i)/n, i = 0, \ldots, n$ and $d_i = i/n, i = 1, \ldots, n$. Try different values for n.

4. Suppose the following chemical reaction occurs inside a volume V:

$$X \underset{c_2}{\overset{c_1}{\rightleftharpoons}} Y\,, \quad 2X \underset{c_4}{\overset{c_3}{\rightleftharpoons}} Z\,, \quad W + X \underset{c_6}{\overset{c_5}{\rightleftharpoons}} 2X\,,$$

where $c_1, \ldots, c_6$ are volume independent *reaction rates*. This stochastic chemical reaction can be modelled as a four-dimensional Markov jump

process with system state at time t represented by the vector $\boldsymbol{J}_V(t) = (w_V(t), x_V(t), y_V(t), z_V(t))^\top$. The transition rates are:

$$\begin{aligned}
q_V((w,x,y,z),(w,x-1,y+1,z)) &= c_1\, x \\
q_V((w,x,y,z),(w,x+1,y-1,z)) &= c_2\, y \\
q_V((w,x,y,z),(w,x-2,y,z+1)) &= c_3\, x(x-1)/(2V) \\
q_V((w,x,y,z),(w,x+2,y,z-1)) &= c_4\, z \\
q_V((w,x,y,z),(w-1,x+1,y,z)) &= c_5\, w\, x/V \\
q_V((w,x,y,z),(w+1,x-1,y,z)) &= c_6\, x(x-1)/(2V)\,.
\end{aligned}$$

(a) Explain why, reasoning probabilistically, the transition rates should be as given above.

(b) Simulate the Markov jump process $\{\boldsymbol{J}_V(t)\}$ under rates $c_1 = c_2 = c_3 = c_4 = c_5 = 1$ and $c_6 = 10$, and initial concentrations of $w_V(0) = x_V(0) = y_V(0) = z_V(0) = 100$. Draw realizations of the processes.

(c) What happens if we take initial concentrations $w_V(0) = x_V(0) = y_V(0) = z_V(0) = 100V$ and let V get larger and larger?

5. Implement the Ornstein–Uhlenbeck process via Euler's method. Take $\nu = 1$, $\theta = 2$, and $\sigma = 0.2$. Try different starting positions, and take 10^6 steps with a step size of 10^{-4}.

Chapter 5

Markov Chain Monte Carlo

The only good Monte Carlo is a dead Monte Carlo.

Trotter and Tukey

Markov chain Monte Carlo (MCMC) is a generic method for *approximate* sampling from an arbitrary distribution. The main idea is to generate a Markov chain whose limiting distribution is equal to the desired distribution.

For *exact* methods for random variable generation from commonly used distributions, see Chapter 2. Applications of MCMC to optimization can be found in Chapter 8, in particular Section 8.3, which discusses *simulated annealing*.

☞ 25
☞ 145

5.1 Metropolis–Hastings Algorithm

The MCMC method originates from Metropolis et al. and applies to the following setting. Suppose that we wish to generate samples from an arbitrary multidimensional pdf

$$f(\mathbf{x}) = \frac{p(\mathbf{x})}{\mathcal{Z}}, \quad \mathbf{x} \in \mathscr{X} \ ,$$

where $p(\mathbf{x})$ is a known positive function and $\mathcal{Z}$ is a known or unknown normalizing constant. Let $q(\mathbf{y} \,|\, \mathbf{x})$ be a **proposal** or **instrumental** density: a Markov transition density describing how to go from state $\mathbf{x}$ to $\mathbf{y}$. Similar to the acceptance–rejection method, the Metropolis–Hastings algorithm is based on the following "trial-and-error" strategy.

☞ 38

Algorithm 5.1 (Metropolis–Hastings Algorithm) *To sample from a density $f(\mathbf{x})$ known up to a normalizing constant, initialize with some $\mathbf{X}_0$ for which $f(\mathbf{X}_0) > 0$. Then, for each $t = 0, 1, 2, \ldots, T-1$ execute the following steps:*

1. *Given the current state $\mathbf{X}_t$, generate $\mathbf{Y} \sim q(\mathbf{y} \,|\, \mathbf{X}_t)$.*

2. *Generate $U \sim \mathsf{U}(0,1)$ and deliver*

$$\mathbf{X}_{t+1} = \begin{cases} \mathbf{Y} & \text{if } \ U \leqslant \alpha(\mathbf{X}_t, \mathbf{Y}) \\ \mathbf{X}_t & \text{otherwise} \ , \end{cases} \tag{5.1}$$

where

$$\alpha(\mathbf{x},\mathbf{y}) = \min\left\{\frac{f(\mathbf{y})\,q(\mathbf{x}\,|\,\mathbf{y})}{f(\mathbf{x})\,q(\mathbf{y}\,|\,\mathbf{x})}, 1\right\}\ . \tag{5.2}$$

The probability $\alpha(\mathbf{x},\mathbf{y})$ is called the **acceptance probability**. Note that in (5.2) we may replace f by p.

We thus obtain the so-called **Metropolis–Hastings Markov chain**, $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_T$, with $\mathbf{X}_T$ approximately distributed according to $f(\mathbf{x})$ for large T. A single Metropolis–Hastings iteration is equivalent to generating a point from the transition density $\kappa(\mathbf{x}_{t+1}\,|\,\mathbf{x}_t)$, where

$$\kappa(\mathbf{y}\,|\,\mathbf{x}) = \alpha(\mathbf{x},\mathbf{y})\,q(\mathbf{y}\,|\,\mathbf{x}) + (1-\alpha^*(\mathbf{x}))\,\delta_{\mathbf{x}}(\mathbf{y})\ , \tag{5.3}$$

with $\alpha^*(\mathbf{x}) = \int \alpha(\mathbf{x},\mathbf{y})\,q(\mathbf{y}\,|\,\mathbf{x})\,\mathrm{d}\mathbf{y}$ and $\delta_{\mathbf{x}}(\mathbf{y})$ denoting the Dirac delta function. Since

$$f(\mathbf{x})\,\alpha(\mathbf{x},\mathbf{y})\,q(\mathbf{y}\,|\,\mathbf{x}) = f(\mathbf{y})\,\alpha(\mathbf{y},\mathbf{x})\,q(\mathbf{x}\,|\,\mathbf{y})$$

and

$$(1-\alpha^*(\mathbf{x}))\,\delta_{\mathbf{x}}(\mathbf{y})\,f(\mathbf{x}) = (1-\alpha^*(\mathbf{y}))\,\delta_{\mathbf{y}}(\mathbf{x})\,f(\mathbf{y})\ ,$$

the transition density satisfies the *detailed balance equation*:

$$f(\mathbf{x})\,\kappa(\mathbf{y}\,|\,\mathbf{x}) = f(\mathbf{y})\,\kappa(\mathbf{x}\,|\,\mathbf{y})\ ,$$

from which it follows that f is the stationary pdf of the chain. In addition, if the transition density q satisfies the conditions

$$\mathbb{P}(\alpha(\mathbf{X}_t,\mathbf{Y}) < 1\,|\,\mathbf{X}_t) > 0\ ,$$

that is, the event $\{\mathbf{X}_{t+1} = \mathbf{X}_t\}$ has positive probability, and

$$q(\mathbf{y}\,|\,\mathbf{x}) > 0 \quad \text{for all } \mathbf{x},\mathbf{y} \in \mathscr{X}\,,$$

then f is the limiting pdf of the chain. As a consequence, to estimate an expectation $\mathbb{E}H(\mathbf{X})$, with $\mathbf{X} \sim f$, one can use the following **ergodic** estimator:

$$\frac{1}{T+1}\sum_{t=0}^{T} H(\mathbf{X}_t)\ . \tag{5.4}$$

The original Metropolis algorithm is suggested for symmetric proposal functions; that is, for $q(\mathbf{y}\,|\,\mathbf{x}) = q(\mathbf{x}\,|\,\mathbf{y})$. Hastings modified the original MCMC algorithm to allow non-symmetric proposal functions, hence the name Metropolis–Hastings algorithm.

5.1.1 Independence Sampler

If the proposal function $q(\mathbf{y}\,|\,\mathbf{x})$ does not depend on $\mathbf{x}$, that is, $q(\mathbf{y}\,|\,\mathbf{x}) = g(\mathbf{y})$ for some pdf $g(\mathbf{y})$, then the acceptance probability is

$$\alpha(\mathbf{x},\mathbf{y}) = \min\left\{\frac{f(\mathbf{y})\,g(\mathbf{x})}{f(\mathbf{x})\,g(\mathbf{y})},\ 1\right\},$$

and Algorithm 5.1 is referred to as the **independence sampler**. The independence sampler is very similar to the acceptance–rejection method in Chapter 2. Just as in that method, it is important that the proposal density g is close to the target f. Note, however, that in contrast to the acceptance–rejection method the independence sampler produces *dependent* samples. In addition, if there is a constant C such that

$$f(\mathbf{x}) = \frac{p(\mathbf{x})}{\int p(\mathbf{x})\,\mathrm{d}\mathbf{x}} \leqslant C g(\mathbf{x})$$

for all $\mathbf{x}$, then the acceptance rate in (5.1) is at least $1/C$ whenever the chain is in stationarity; namely,

$$\begin{aligned}
\mathbb{P}(U \leqslant \alpha(\mathbf{X},\mathbf{Y})) &= \iint \min\left\{\frac{f(\mathbf{y})\,g(\mathbf{x})}{f(\mathbf{x})\,g(\mathbf{y})},\,1\right\} f(\mathbf{x})\,g(\mathbf{y})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \\
&= 2\iint \mathrm{I}\left\{\frac{f(\mathbf{y})\,g(\mathbf{x})}{f(\mathbf{x})\,g(\mathbf{y})} \geqslant 1\right\} f(\mathbf{x})\,g(\mathbf{y})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \\
&\geqslant \frac{2}{C}\iint \mathrm{I}\left\{\frac{f(\mathbf{y})\,g(\mathbf{x})}{f(\mathbf{x})\,g(\mathbf{y})} \geqslant 1\right\} f(\mathbf{x})\,f(\mathbf{y})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \\
&\geqslant \frac{2}{C}\,\mathbb{P}\left(\frac{f(\mathbf{Y})}{g(\mathbf{Y})} \geqslant \frac{f(\mathbf{X})}{g(\mathbf{X})}\right) = \frac{1}{C}\,.
\end{aligned}$$

In contrast, the acceptance rate in Algorithm 2.8 (acceptance–rejection) using g as a proposal is always equal to $1/C$. ☞ 38

5.1.2 Random Walk Sampler

If the proposal is symmetric, that is, $q(\mathbf{y}\,|\,\mathbf{x}) = q(\mathbf{x}\,|\,\mathbf{y})$, then the acceptance probability (5.2) is

$$\alpha(\mathbf{x},\mathbf{y}) = \min\left\{\frac{f(\mathbf{y})}{f(\mathbf{x})},\,1\right\}, \tag{5.5}$$

and Algorithm 5.1 is referred to as the **random walk sampler**. An example of a random walk sampler is when $\mathbf{Y} = \mathbf{X}_t + \sigma\mathbf{Z}$ in Step 1 of Algorithm 5.1, where $\mathbf{Z}$ is typically generated from some spherically symmetrical distribution (in the continuous case), such as $\mathsf{N}(\mathbf{0}, I)$.

Example 5.1 (Bayesian Analysis of the Logit Model) Consider the Bayesian analysis of the logistic regression model or **logit model**. This is a commonly used generalized linear model, where binary data $y_1,\ldots,y_n$ (the responses) are assumed to be conditionally independent realizations from $\mathsf{Ber}(p_i)$ given $p_1,\ldots,p_n$ (that is, $y_i\,|\,p_i \sim \mathsf{Ber}(p_i)$, $i = 1,\ldots,n$, independently), with

$$p_i = \frac{1}{1+\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}}}\,, \quad i = 1,\ldots,n\,.$$

Here, $\mathbf{x}_i = (x_{i1}, x_{i2},\ldots,x_{ik})^\top$ are the explanatory variables or covariates for the i-th response and $\boldsymbol{\beta} = (\beta_1,\ldots,\beta_k)^\top$ are the parameters of the model with multivariate normal prior: $\mathsf{N}(\boldsymbol{\beta}_0, V_0)$. Thus, the Bayesian logit model can be summarized as:

- Prior: $f(\boldsymbol{\beta}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)^\top V_0^{-1}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)\right),\ \boldsymbol{\beta} \in \mathbb{R}^k$.
- Likelihood: $f(\mathbf{y}\,|\,\boldsymbol{\beta}) = \prod_{i=1}^n p_i^{y_i}(1-p_i)^{1-y_i},\ p_i^{-1} = 1+\exp(-\mathbf{x}_i^\top\boldsymbol{\beta})$.

Since the posterior pdf $f(\boldsymbol{\beta}\,|\,\mathbf{y}) \propto f(\boldsymbol{\beta},\mathbf{y}) = f(\boldsymbol{\beta})f(\mathbf{y}\,|\,\boldsymbol{\beta})$ cannot be written in a simple analytical form, the Bayesian analysis proceeds by (approximately) drawing a sample from the posterior $f(\boldsymbol{\beta}\,|\,\mathbf{y})$ in order to obtain estimates of various quantities of interest such as $\mathbb{E}[\boldsymbol{\beta}\,|\,\mathbf{y}]$ and $\mathrm{Cov}(\boldsymbol{\beta}\,|\,\mathbf{y})$. In addition, simulation allows a convenient way to explore the marginal posterior densities of each model parameter.

To approximately draw from the posterior we use the random walk sampler
☞ 58
with a multivariate $\mathbf{t}_\nu(\boldsymbol{\mu},\Sigma)$ proposal tailored to match the overall shape of the posterior around its mode. The vector $\boldsymbol{\mu}$ is taken as the mode of the posterior; that is, as $\mathrm{argmax}_{\boldsymbol{\beta}} \ln f(\boldsymbol{\beta}\,|\,\mathbf{y})$, which can be obtained approximately via a Newton–Raphson procedure with gradient

$$\nabla \ln f(\boldsymbol{\beta}\,|\,\mathbf{y}) = \sum_{i=1}^n \left(y_i - \frac{1}{1+\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}}}\right)\mathbf{x}_i - V_0^{-1}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)\,,$$

and Hessian

$$H = -\sum_{i=1}^n \frac{\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}}}{(1+\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}})^2}\,\mathbf{x}_i\,\mathbf{x}_i^\top - V_0^{-1}\,,$$

where we have used the fact that the logarithm of the posterior (ignoring constant terms) is:

$$-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\beta}_0)^\top V_0^{-1}(\boldsymbol{\beta}-\boldsymbol{\beta}_0) - \sum_{i=1}^n y_i \ln\left(1+\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}}\right) + (1-y_i)\left(\mathbf{x}_i^\top\boldsymbol{\beta} + \ln\left(1+\mathrm{e}^{-\mathbf{x}_i^\top\boldsymbol{\beta}}\right)\right)\,.$$

☞ 58
The scale matrix Σ of the proposal distribution $\mathbf{t}_\nu(\boldsymbol{\mu},\Sigma)$ is chosen as the inverse of the observed Fisher information matrix: $\Sigma = -H^{-1}$. Finally, the shape ν (degrees of freedom) is arbitrarily set to 10. The random walk sampler is initialized at the mode $\boldsymbol{\mu}$ of the posterior. If $\boldsymbol{\beta}^*$ is a newly generated proposal and $\boldsymbol{\beta}$ is the current value, the acceptance criterion (5.2) in the Metropolis–Hastings algorithm can be written as:

$$\alpha(\boldsymbol{\beta},\boldsymbol{\beta}^*) = \min\left\{\frac{f(\boldsymbol{\beta}^*,\mathbf{y})}{f(\boldsymbol{\beta},\mathbf{y})}, 1\right\}.$$

The following MATLAB code implements this procedure for the logit model using an artificial data set.

```
%logit_model.m
clear all,clc
n=5000; % number of data points (y_1,...,y_n)
k=3;   % number of explanatory variables
% generate artificial dataset
randn('seed', 12345);  rand('seed', 67890);
truebeta = [1 -5.5 1]';
```

```
X = [ ones(n,1) randn(n,k-1)*0.1 ];  % design matrix
Y = binornd(1,1./(1+exp(-X*truebeta)));
bo=zeros(k,1); % we set Vo=100*eye(k);
% determine the mode using Newton Raphson
err=inf; b=bo; % initial guess
while norm(err)>10^(-3)
    p=1./(1+exp(-X*b));
    g=X'*(Y-p)-(b-bo)/100;
    H=-X'*diag(p.^2.*(1./p-1))*X-eye(k)/100;
    err=H\g; % compute Newton-Raphson correction
    b=b-err; % update Newton guess
end
% scale parameter for proposal
Sigma=-H\eye(k); B=chol(Sigma);
% logarithm of joint density (up to a constant)
logf=@(b)(-.5*(b-bo)'*(b-bo)/100-Y'*log(1+exp(-X*b))...
    -(1-Y)'*(X*b+log(1+exp(-X*b))));
alpha=@(x,y)min(1,exp(logf(y)-logf(x)));
df=10; T=10^4; data=nan(T,k); %allocate memory
for t=1:T
    % make proposal from multivariate t
    b_star= b + B*(sqrt(df/gamrnd(df/2,2))*randn(k,1));
    if rand<alpha(b,b_star)
        b=b_star;
    end
    data(t,:)=b';
end
b_hat=mean(data)
Cov_hat=cov(data)
```

Typical estimates for the posterior mean $\mathbb{E}[\boldsymbol{\beta}\,|\,\mathbf{y}]$ and covariance $\mathrm{Cov}(\boldsymbol{\beta}\,|\,\mathbf{y})$ are

$$\widehat{\mathbb{E}[\boldsymbol{\beta}\,|\,\mathbf{y}]} = \begin{pmatrix} 0.980 \\ -5.313 \\ 1.136 \end{pmatrix} \text{ and } \widehat{\mathrm{Cov}(\boldsymbol{\beta}\,|\,\mathbf{y})} = \begin{pmatrix} 0.0011 & -0.0025 & 0.0005 \\ -0.0025 & 0.1116 & -0.0095 \\ 0.0005 & -0.0095 & 0.1061 \end{pmatrix}.$$

Since the prior we employed is relatively non-informative, it is not surprising that the estimated posterior mean is very similar to the maximum likelihood estimate: $\widehat{\boldsymbol{\beta}} = (0.978, -5.346, 1.142)^\top$.

5.2 Gibbs Sampler

The **Gibbs sampler** can be viewed as a particular instance of the Metropolis–Hastings algorithm for generating n-dimensional random vectors. Due to its importance it is presented separately. The distinguishing feature of the Gibbs

sampler is that the underlying Markov chain is constructed from a sequence of conditional distributions, in either a deterministic or random fashion.

Suppose that we wish to sample a random vector $\mathbf{X} = (X_1, \ldots, X_n)$ according to a target pdf $f(\mathbf{x})$. Let $f(x_i \,|\, x_1, \ldots, x_{i-1},\ x_{i+1}, \ldots, x_n)$ represent the conditional pdf of the i-th component, X_i, given the other components $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. Here we use a Bayesian notation.

Algorithm 5.2 (Gibbs Sampler) *Given an initial state $\mathbf{X}_0$, iterate the following steps for $t = 0, 1, \ldots$.*

1. *For a given $\mathbf{X}_t$, generate $\mathbf{Y} = (Y_1, \ldots, Y_n)$ as follows:*
   (a) *Draw Y_1 from the conditional pdf $f(x_1 \,|\, X_{t,2}, \ldots, X_{t,n})$.*
   (b) *Draw Y_i from $f(x_i \,|\, Y_1, \ldots, Y_{i-1}, X_{t,i+1}, \ldots, X_{t,n})$, $\quad i = 2, \ldots, n-1$.*
   (c) *Draw Y_n from $f(x_n \,|\, Y_1, \ldots, Y_{n-1})$.*
2. *Let $\mathbf{X}_{t+1} = \mathbf{Y}$.*

The transition pdf is given by

$$\kappa_{1\to n}(\mathbf{y} \,|\, \mathbf{x}) = \prod_{i=1}^{n} f(y_i \,|\, y_1, \ldots, y_{i-1}, x_{i+1}, \ldots, x_n)\ , \tag{5.6}$$

where the subscript $1 \to n$ indicates that the components of vector $\mathbf{x}$ are updated in the order $1 \to 2 \to 3 \to \cdots \to n$. Note that in the Gibbs sampler *every* "proposal" $\mathbf{y}$, is accepted. The transition density of the reverse move $\mathbf{y} \to \mathbf{x}$, in which the vector $\mathbf{y}$ is updated in the order $n \to n-1 \to n-2 \to \cdots \to 1$ is

$$\kappa_{n\to 1}(\mathbf{x} \,|\, \mathbf{y}) = \prod_{i=1}^{n} f(x_i \,|\, y_1, \ldots, y_{i-1}, x_{i+1}, \ldots, x_n)\ .$$

Hammersley and Clifford prove the following result.

Theorem 5.2.1 (Hammersley–Clifford) *Let $f(x_i)$ be the i-th marginal density of the pdf $f(\mathbf{x})$. Suppose that density $f(\mathbf{x})$ satisfies the* **positivity condition***, that is, for every $\mathbf{y} \in \{\mathbf{x} : f(x_i) > 0,\ i = 1, \ldots, n\}$, we have $f(\mathbf{y}) > 0$. Then,*

$$f(\mathbf{y})\, \kappa_{n\to 1}(\mathbf{x} \,|\, \mathbf{y}) = f(\mathbf{x})\, \kappa_{1\to n}(\mathbf{y} \,|\, \mathbf{x})\ .$$

Proof (outline): Observe that

$$\begin{aligned}
\frac{\kappa_{1\to n}(\mathbf{y} \,|\, \mathbf{x})}{\kappa_{n\to 1}(\mathbf{x} \,|\, \mathbf{y})} &= \prod_{i=1}^{n} \frac{f(y_i \,|\, y_1, \ldots, y_{i-1}, x_{i+1}, \ldots, x_n)}{f(x_i \,|\, y_1, \ldots, y_{i-1}, x_{i+1}, \ldots, x_n)} \\
&= \prod_{i=1}^{n} \frac{f(y_1, \ldots, y_i, x_{i+1}, \ldots, x_n)}{f(y_1, \ldots, y_{i-1}, x_i, \ldots, x_n)} \\
&= \frac{f(\mathbf{y}) \prod_{i=1}^{n-1} f(y_1, \ldots, y_i, x_{i+1}, \ldots, x_n)}{f(\mathbf{x}) \prod_{j=2}^{n} f(y_1, \ldots, y_{j-1}, x_j, \ldots, x_n)} \\
&= \frac{f(\mathbf{y}) \prod_{i=1}^{n-1} f(y_1, \ldots, y_i, x_{i+1}, \ldots, x_n)}{f(\mathbf{x}) \prod_{j=1}^{n-1} f(y_1, \ldots, y_j, x_{j+1}, \ldots, x_n)} = \frac{f(\mathbf{y})}{f(\mathbf{x})}\ .
\end{aligned}$$

The result follows by rearranging the last identity.

The Hammersley–Clifford condition is similar to the detailed balance condition for the Metropolis–Hastings sampler, because integrating both sides with respect to $\mathbf{x}$ yields the global balance equation:

$$\int f(\mathbf{x})\, \kappa_{1\to n}(\mathbf{y} \,|\, \mathbf{x})\, \mathrm{d}\mathbf{y} = f(\mathbf{y}) \;,$$

from which we can conclude that f is the stationary pdf of the Markov chain with transition density $\kappa_{1\to n}(\mathbf{y} \,|\, \mathbf{x})$. In addition, it can be shown that the positivity assumption on f implies that the Gibbs Markov chain is irreducible and that f is its limiting pdf. In practice the positivity condition is difficult to verify. However, there are a number of weaker and more technical conditions which ensure that the limiting pdf of the process $\{\mathbf{X}_t, t = 1, 2, \ldots\}$ generated via the Gibbs sampler is f, and that the convergence to f is geometrically fast.

Algorithm 5.2 presents a **systematic** (coordinatewise) Gibbs sampler. That is, the components of vector $\mathbf{X}$ are updated in the coordinatewise order $1 \to 2 \to \cdots \to n$. The completion of all the conditional sampling steps in the specified order is called a **cycle**. Alternative updating of the components of vector $\mathbf{X}$ are possible. In the **reversible Gibbs sampler** a single cycle consists of the coordinatewise updating

$$1 \to 2 \to \cdots \to n-1 \to n \to n-1 \to \cdots \to 2 \to 1 \;.$$

In the **random sweep/scan Gibbs sampler** a single cycle can either consist of one or several coordinates selected uniformly from the integers $1, \ldots, n$, or a random permutation $\pi_1 \to \pi_2 \to \cdots \to \pi_n$ of all coordinates. In all cases, except for the systematic Gibbs sampler, the resulting Markov chain $\{\mathbf{X}_t, t = 1, 2, \ldots\}$ is *reversible*. In the case where a cycle consists of a single randomly selected coordinate, the random Gibbs sampler can be formally viewed as a Metropolis–Hastings sampler with transition function

$$q(\mathbf{y} \,|\, \mathbf{x}) = \frac{1}{n}\, f(y_i \,|\, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \frac{1}{n}\, \frac{f(\mathbf{y})}{\sum_{y_i} f(\mathbf{y})} \;,$$

where $\mathbf{y} = (x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_n)$. Since $\sum_{y_i} f(\mathbf{y})$ can also be written as $\sum_{x_i} f(\mathbf{x})$, we have

$$\frac{f(\mathbf{y})\, q(\mathbf{x} \,|\, \mathbf{y})}{f(\mathbf{x})\, q(\mathbf{y} \,|\, \mathbf{x})} = \frac{f(\mathbf{y})\, f(\mathbf{x})}{f(\mathbf{x})\, f(\mathbf{y})} = 1 \;,$$

so that the acceptance probability $\alpha(\mathbf{x}, \mathbf{y})$ is 1 in this case.

Example 5.2 (Zero-Inflated Poisson Model) Gibbs sampling is one of the main computational techniques used in Bayesian analysis. In the **zero-inflated Poisson** model, the random data $X_1, \ldots, X_n$ are assumed to be of the form $X_i = R_i\, Y_i$, where the $Y_1, \ldots, Y_n \sim_{\text{iid}} \mathsf{Poi}(\lambda)$ are independent of $R_1, \ldots, R_n \sim_{\text{iid}} \mathsf{Ber}(p)$. Given an outcome $\mathbf{x} = (x_1, \ldots, x_n)$, the objective is to estimate both λ and p. A typical Bayesian data analysis gives the following hierarchical model:

- $p \sim \mathsf{U}(0,1)$ (prior for p),
- $(\lambda \,|\, p) \sim \mathsf{Gamma}(a,b)$ (prior for λ),
- $(r_i \,|\, p,\lambda) \sim \mathsf{Ber}(p)$ independently (from the model above),
- $(x_i \,|\, \mathbf{r},\lambda,p) \sim \mathsf{Poi}(\lambda\, r_i)$ independently (from the model above),

where a and b are known parameters. It follows that the joint pdf of all parameters and $\mathbf{x}$ is

$$\begin{aligned} f(\mathbf{x},\mathbf{r},\lambda,p) &= \frac{b^a \lambda^{a-1} \mathrm{e}^{-b\lambda}}{\Gamma(a)} \prod_{i=1}^{n} \frac{\mathrm{e}^{-\lambda r_i} (\lambda\, r_i)^{x_i}}{x_i!}\, p^{r_i} (1-p)^{1-r_i} \\ &= \frac{b^a \lambda^{a-1} \mathrm{e}^{-b\lambda}}{\Gamma(a)}\, \mathrm{e}^{-\lambda \sum_i r_i}\, p^{\sum_i r_i}\, (1-p)^{n-\sum_i r_i}\, \lambda^{\sum_i x_i} \prod_{i=1}^{n} \frac{r_i^{x_i}}{(x_i)!}\,. \end{aligned}$$

The posterior pdf $f(\lambda, p, \mathbf{r} \,|\, \mathbf{x}) \propto f(\mathbf{x},\mathbf{r},\lambda,p)$ is of large dimension, which makes analytical computation using Bayes' formula intractable. Instead, the Gibbs sampler (Algorithm 5.2) provides a convenient tool for approximate sampling and exploration of the posterior. Here the conditionals of the posterior are:

- $f(\lambda \,|\, p,\mathbf{r},\mathbf{x}) \propto \lambda^{a-1+\sum_i x_i}\, \mathrm{e}^{-\lambda(b+\sum_i r_i)}$,
- $f(p \,|\, \lambda,\mathbf{r},\mathbf{x}) \propto p^{\sum_i r_i}\, (1-p)^{n-\sum_i r_i}$,
- $f(r_k \,|\, \lambda,p,\mathbf{x}) \propto \left(\dfrac{p\,\mathrm{e}^{-\lambda}}{1-p}\right)^{r_k} r_k^{x_k}\,.$

In other words, we have:

- $(\lambda \,|\, p,\mathbf{r},\mathbf{x}) \sim \mathsf{Gamma}\Big(a + \sum_i x_i,\; b + \sum_i r_i\Big)$,
- $(p \,|\, \lambda,\mathbf{r},\mathbf{x}) \sim \mathsf{Beta}\Big(1 + \sum_i r_i,\; n + 1 - \sum_i r_i\Big)$,
- $(r_k \,|\, \lambda,p,\mathbf{x}) \sim \mathsf{Ber}\left(\dfrac{p\,\mathrm{e}^{-\lambda}}{p\,\mathrm{e}^{-\lambda} + (1-p)\mathrm{I}_{\{x_k=0\}}}\right)$.

To test the accuracy of the Gibbs sampler (Algorithm 5.2), we generate $n = 100$ random data points from the zero-inflated Poisson model using parameters $p = 0.3$ and $\lambda = 2$. To recover the parameters from the data, we choose $a = 1$ and $b = 1$ for the prior distribution of λ, and generate a (dependent) sample of size 10^5 from the posterior distribution using the Gibbs sampler. A 95% Bayesian confidence interval (credible interval) is constructed using the script below. Note that MATLAB's *statistics toolbox* function `gamrnd(a,b)` draws from the $\mathsf{Gamma}(a, 1/b)$ distribution. The estimated Bayesian confidence intervals are $(1.33, 2.58)$ for λ and $(0.185, 0.391)$ for p. Observe that the true values lie within these intervals.

```
%zip.m
n=100; p=.3; lambda=2;
% generate ZIP random variables
```

```
data=poissrnd(lambda,n,1).*(rand(n,1)<p);
% now try to recover the ZIP parameters from the data
P=rand; % starting guess for p
lam=gamrnd(1,1); % starting guess for lambda
r=(rand(n,1)<P); % starting guess for  r
Sum_data=sum(data);
gibbs_sample=zeros(10^5,2);
% apply the Gibbs sampler
for k=1:10^5
   Sum_r=sum(r);
   lam=gamrnd(1+Sum_data,1/(1+Sum_r));
   P=betarnd(1+Sum_r,n+1-Sum_r);
   prob=exp(-lam)*P./(exp(-lam)*P+(1-P)*(data==0));
   r=(rand(n,1)<prob);
   gibbs_sample(k,:)=[P,lam];
end
% 95% probability interval for lambda
prctile(gibbs_sample(:,2),[2.5,97.5])
% 95% probability interval for p
prctile(gibbs_sample(:,1),[2.5,97.5])
```

Gibbs sampling is advantageous whenever it is easy to sample from the conditional distributions of the joint density. Note that it is not necessary to update each component of the random vector $\mathbf{X}$ individually. Instead, blocks or groups of variables can be updated simultaneously. For example, to sample from the joint pdf $f(x_1, x_2, x_3)$ we can consider the following version of the Gibbs sampler.

Algorithm 5.3 (Grouped Gibbs Sampler) *To sample from* $f(x_1, x_2, x_3)$ *with a given initial state* $\mathbf{X}_0$*, iterate the following steps for* $t = 0, 1, 2, \ldots$.

1. *For a given* $\mathbf{X}_t = (X_{t,1}, X_{t,2}, X_{t,3})$*, generate* $\mathbf{Y} = (Y_1, Y_2, Y_3)$ *as follows:*
   (a) *Draw* (Y_1, Y_2) *from the conditional pdf* $f(y_1, y_2 \,|\, X_{t,3})$.
   (b) *Draw* Y_3 *from the conditional pdf* $f(y_3 \,|\, Y_1, Y_2)$.
2. *Let* $\mathbf{X}_{t+1} = \mathbf{Y}$.

The grouped variables in Algorithm 5.3 are x_1 and x_2. Significant speed-up of the convergence of the chain can be achieved when highly correlated variables are grouped together.

5.3 Hit-and-Run Sampler

The **hit-and-run** sampler, pioneered by Smith, is among the first MCMC samplers in the category of **line samplers**.. As in previous sections, the objective is to sample from a target distribution $f(\mathbf{x}) = p(\mathbf{x})/\mathcal{Z}$ on $\mathscr{X} \subseteq \mathbb{R}^n$.

We first describe the original hit-and-run sampler for generating from a *uniform* distribution on a bounded open region $\mathscr{X}$ of $\mathbb{R}^n$. At each iteration, starting from a current point $\mathbf{x}$, a **direction vector d** is generated uniformly on the surface of an n-dimensional hypersphere. The intersection of the corresponding bidirectional line through $\mathbf{x}$ and the enclosing box of $\mathscr{X}$ defines a line segment $\mathscr{L}$. The next point $\mathbf{y}$ is then selected uniformly from the intersection of $\mathscr{L}$ and $\mathscr{X}$.

Figure 5.1 illustrates the hit-and-run algorithm for generating uniformly from the set $\mathscr{X}$ (the gray region) which is bounded by a rectangle. Given the point $\mathbf{x}$ in $\mathscr{X}$, the direction $\mathbf{d}$ is generated, which defines the line segment $\mathscr{L} = uv$. Then, a point $\mathbf{y}$ is chosen uniformly on $\mathscr{L} \cap \mathscr{X}$, for example, by the acceptance–rejection method, that is, generate a point uniformly on $\mathscr{L}$ and then accept this point only if it lies in $\mathscr{X}$.

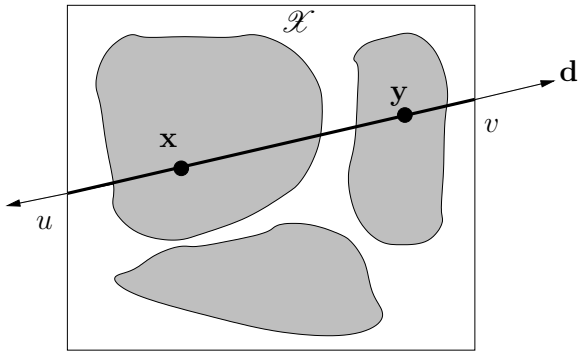


Figure 5.1: Illustration of hit-and-run on a square in two dimensions.

The uniform hit-and-run sampler asymptotically generates uniformly distributed points over *arbitrary* open regions of $\mathbb{R}^n$. One desirable property of hit-and-run is that it can globally reach any point in the set in one step; that is, there is a positive probability of sampling any neighborhood in the set. Lovász proves that hit-and-run on a convex body in n dimensions produces an approximately uniformly distributed sample in polynomial time, $\mathcal{O}(n^3)$. He notes that in practice the hit-and-run algorithm appears to offer the most rapid convergence to a uniform distribution.

We now describe a more general version of the hit-and-run algorithm for sampling from a *any* strictly positive continuous pdf $f(\mathbf{x}) = p(\mathbf{x})/\mathscr{Z}$ on any region $\mathscr{X}$ — bounded or unbounded. Similar to the Metropolis–Hastings algorithm we generate a proposal move, which is then accepted or rejected with probability that depends on f. A proposal move $\mathbf{y}$ is generated by stepping away from the current point (at iteration t) $\mathbf{x}$ in the direction $\mathbf{d}$ with a step of size λ. The step size λ at iteration t is generated from a proposal density $g_t(\lambda \,|\, \mathbf{d}, \mathbf{x})$. The candidate point $\mathbf{y} = \mathbf{x} + \lambda\,\mathbf{d}$ is then accepted with probability

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{ \frac{f(\mathbf{y})\, g_t\left(|\lambda| \,\middle|\, -\operatorname{sgn}(\lambda)\,\mathbf{d}, \mathbf{y}\right)}{f(\mathbf{x})\, g_t\left(|\lambda| \,\middle|\, \operatorname{sgn}(\lambda)\,\mathbf{d}, \mathbf{x}\right)},\ 1 \right\}, \tag{5.7}$$

as in the Metropolis–Hastings acceptance criterion (5.1); otherwise, the Markov chain remains in the current state $\mathbf{x}$. The condition (5.7) ensures that g_t satisfies the detailed balance condition:

$$g_t\left(\|\mathbf{x}-\mathbf{y}\|\left|\frac{\mathbf{y}-\mathbf{x}}{\|\mathbf{x}-\mathbf{y}\|},\mathbf{x}\right.\right)\alpha(\mathbf{x},\mathbf{y})f(\mathbf{x}) = g_t\left(\|\mathbf{x}-\mathbf{y}\|\left|\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|},\mathbf{y}\right.\right)\alpha(\mathbf{y},\mathbf{x})f(\mathbf{y}) \; .$$

We refer to proposal densities that satisfy the detailed balance equations as **valid** proposals. At iteration t, let

$$\mathscr{M}_t \stackrel{\text{def}}{=} \{\lambda \in \mathbb{R} : \; \mathbf{x} + \lambda\,\mathbf{d} \in \mathscr{X}\} \; .$$

A valid proposal density $g_t(\lambda \,|\, \mathbf{d}, \mathbf{x})$ can be one of the following:

- $g_t(\lambda \,|\, \mathbf{d}, \mathbf{x}) = \widetilde{g}_t(\mathbf{x} + \lambda\,\mathbf{d}), \;\; \lambda \in \mathbb{R}$. The acceptance probability (5.7) then simplifies to
$$\alpha(\mathbf{x},\mathbf{y}) = \min\left\{\frac{f(\mathbf{y})\,\widetilde{g}_t(\mathbf{x})}{f(\mathbf{x})\,\widetilde{g}_t(\mathbf{y})}, 1\right\} .$$
A common choice is
$$g_t(\lambda \,|\, \mathbf{d}, \mathbf{x}) = \frac{f(\mathbf{x}+\lambda\,\mathbf{d})}{\int_{\mathscr{M}_t} f(\mathbf{x}+u\,\mathbf{d})\,\mathrm{d}u}, \quad \lambda \in \mathscr{M}_t \; , \tag{5.8}$$
in which case (5.7) further simplifies to $\alpha(\mathbf{x},\mathbf{y}) = 1$.

- $g_t(\lambda \,|\, \mathbf{d}, \mathbf{x}) = \widetilde{g}_t(\lambda), \;\; \lambda \in \mathbb{R}$, is a symmetric ($\widetilde{g}_t(\lambda) = \widetilde{g}_t(-\lambda)$) continuous pdf that may depend only on $\mathscr{M}_t$. The acceptance probability (5.7) then simplifies to
$$\alpha(\mathbf{x},\mathbf{y}) = \min\{f(\mathbf{y})/f(\mathbf{x}), 1\} \; .$$
If $\mathscr{X}$ is unbounded it is common to choose $\widetilde{g}_t(\lambda)$ as the normal pdf with mean 0 and variance that depends on $\mathscr{M}_t$. Alternatively, if $\mathscr{X}$ is bounded a common choice is
$$\widetilde{g}_t(\lambda) = \frac{\mathrm{I}_{\{\lambda\in\mathscr{M}_t\}}}{\int_{\mathbb{R}} \mathrm{I}_{\{u\in\mathscr{M}_t\}}\,\mathrm{d}u} \; .$$

In summary, the hit-and-run algorithm reads as follows.

Algorithm 5.4 (Hit-and-Run)

1. *Initialize with $\mathbf{X}_1 \in \mathscr{X}$ and set $t = 1$.*
2. *Generate a random direction $\mathbf{d}_t$ according to a uniform distribution on the unit n-dimensional hypersphere. In other words, generate:*
$$\mathbf{d}_t = \left(\frac{Z_1}{\|\mathbf{Z}\|}, \cdots, \frac{Z_n}{\|\mathbf{Z}\|}\right)^\top, \quad Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathsf{N}(0,1) \; ,$$
where $\|\mathbf{Z}\| = \sqrt{Z_1^2 + \cdots + Z_n^2}$.
3. *Generate λ from a valid proposal density $g_t(\lambda \,|\, \mathbf{d}_t, \mathbf{X}_t)$.*

4. Set $\mathbf{Y} = \mathbf{X}_t + \lambda\,\mathbf{d}_t$*, and let:*

$$\mathbf{X}_{t+1} = \begin{cases} \mathbf{Y} & \textit{with probability } \alpha(\mathbf{X}_t, \mathbf{Y}) \textit{ in } (5.7) \\ \mathbf{X}_t & \textit{otherwise.} \end{cases}$$

5. If a stopping criterion is met, stop; otherwise, increment t *and repeat from Step 2.*

Note that the proposal (5.8) gives a Gibbs-type sampling algorithm in which every candidate point is accepted and $\mathbf{X}_{t+1} \neq \mathbf{X}_t$.

Example 5.3 (Truncated Multivariate Normal Generator) A common computational problem in Bayesian data analysis involves sampling from a truncated multivariate normal pdf:

$$f(\mathbf{x}) \propto p(\mathbf{x}) = \exp\Big(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big)\, \mathrm{I}_{\{\mathbf{x}\in\mathscr{X}\}}\,,$$

where $\mathscr{X} \subset \mathbb{R}^n$. The hit-and-run sampler can be used to efficiently sample in such cases. With (5.8) as the proposal, the pdf of λ is:

$$g_t(\lambda \,|\, \mathbf{d}, \mathbf{x}) \propto \exp\left(-\frac{\mathbf{d}^\top \Sigma^{-1}\mathbf{d}}{2}\,\lambda^2 - \mathbf{d}^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\,\lambda \right) \mathrm{I}_{\{\mathbf{x}+\lambda\,\mathbf{d}\in\mathscr{X}\}}\,,$$

which corresponds to a truncated univariate normal distribution with mean

$$-\frac{\mathbf{d}^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}{\mathbf{d}^\top\Sigma^{-1}\mathbf{d}}$$

and variance $(\mathbf{d}^\top\Sigma^{-1}\mathbf{d})^{-1}$, truncated to the set $\mathscr{M}_t = \{\lambda :\ \mathbf{x} + \lambda\,\mathbf{d} \in \mathscr{X}\}$. For example, suppose that $\mathscr{X} = \{\mathbf{x} \in \mathbb{R}^2 : \mathbf{x}^\top\mathbf{x} > 25\}$, then denoting $D = (\mathbf{d}^\top\mathbf{x})^2 - \mathbf{x}^\top\mathbf{x} + 25$, we have two cases:

- $\mathscr{M}_t \equiv \mathbb{R}$, if $D < 0$;
- $\mathscr{M}_t \equiv (-\infty,\ -\sqrt{D} - \mathbf{d}^\top\mathbf{x}] \cup [\sqrt{D} - \mathbf{d}^\top\mathbf{x},\ \infty)$, if $D > 0$.

The following code implements this particular example and generates 10^4 points in $\mathbb{R}^2$ with $\boldsymbol{\mu} = (1/2, 1/2)^\top$ and covariance matrix with $\Sigma_{11} = \Sigma_{22} = 1$, $\Sigma_{12} = 0.9$. Figure 5.2 shows a scatter plot of the output and the boundary of $\mathscr{X}$.

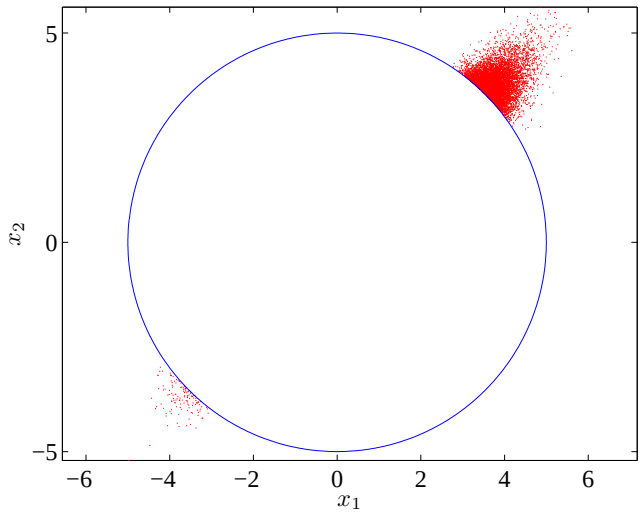


Figure 5.2: Hit-and-run sampling from a truncated bivariate normal density.

```
%Hit_and_run.m
Sig=[1,0.9;0.9,1]; Mu=[1,1]'/2; B=chol(Sig)';
x=[5,5]';              %starting point
T=10^4; data=nan(T,2);% number of points desired
for t=1:T
    d=randn(2,1); d=d/norm(d);
    D=(d'*x)^2-x'*x+25;
    z1=B\d; z2=B\(x-Mu);
    % determine mean and std of lambda dist.
    sig=1/norm(z1); mu=-z1'*z2*sig^2;
    if D<0
        lam=mu+randn*sig;
    else
        lam=normt2(mu,sig,-sqrt(D)-d'*x,sqrt(D)-d'*x);
    end
    x=x+lam*d;
    data(t,:)=x';
end
plot(data(:,1),data(:,2),'r.'),axis equal,
hold on
ezplot('x^2+y^2-25')
```

The code above uses the function `normt2.m`, which draws from the $\mathsf{N}(\mu, \sigma^2)$ distribution truncated on the set $(-\infty, a] \cup [b, \infty)$, via the inverse-transform method.

```
function out=normt2(mu,sig,a,b)
pb=normcdf((b-mu)/sig);
pa=normcdf((a-mu)/sig);
if rand<pa/(pa+1-pb)
      out=mu+sig*norminv(pa*rand(size(mu)));
else
      out=mu+sig*norminv((1-pb)*rand(size(mu))+pb);
end
```

5.4 Exercises

1. Consider a diffusion process defined by the SDE

$$\mathrm{d}\mathbf{X}_t = \frac{1}{2}\nabla \ln f(\mathbf{X}_t)\,\mathrm{d}t + \mathrm{d}\mathbf{W}_t\ ,$$

where f is a probability density and $\nabla \ln f(\mathbf{X}_t)$ denotes the gradient of $\ln f(\mathbf{x})$ evaluated at $\mathbf{X}_t$. Suppose the proposal state $\mathbf{Y}$ in Step 1 of Algorithm 5.1 corresponds to the Euler discretization of this SDE for some step size h:

☞ 76

$$\mathbf{Y} = \mathbf{X}_t + \frac{h}{2}\nabla \ln f(\mathbf{X}_t) + \sqrt{h}\,\mathbf{Z}, \quad \mathbf{Z} \sim \mathsf{N}(\mathbf{0}, I)\ .$$

This gives a **Langevin Metropolis–Hastings** sampler. Implement the sampler to draw $N = 10^5$ dependent samples from the $\mathsf{Gamma}(2,1)$ distribution. Use the `kde.m` program (see Exercise 3 in Section 3.4) to assess how well the estimated pdf fits the true pdf. Investigate how the step size h and the length of the burn-in period affect the fit.

2. Let $X_1, \ldots, X_n$ be a random sample from the $\mathsf{N}(\mu, \sigma^2)$ distribution. Consider the following Bayesian model:

- $f(\mu, \sigma^2) = 1/\sigma^2$;
- $(\mathbf{x}_i \,|\, \mu, \sigma) \sim \mathsf{N}(\mu, \sigma^2)$, $i = 1, \ldots, n$ independently.

Note that the prior for (μ, σ^2) is *improper*. That is, it is not a pdf in itself, but by obstinately applying Bayes' formula, it does yield a proper posterior pdf. In some sense it conveys the least amount of information about μ and σ^2. Let $\mathbf{x} = (x_1, \ldots, x_n)$ represent the data. The posterior pdf is given by

$$f(\mu, \sigma^2 \,|\, \mathbf{x}) = \left(2\,\pi\,\sigma^2\right)^{-n/2} \exp\left\{-\frac{1}{2}\frac{\sum_i (x_i - \mu)^2}{\sigma^2}\right\} \frac{1}{\sigma^2}\ .$$

We wish to sample from this pdf.

(a) Derive the distribution of μ given $(\sigma^2, \mathbf{x})$.

(b) Prove that

$$f(\sigma^2 \,|\, \mu, \mathbf{x}) \propto \frac{1}{(\sigma^2)^{n/2+1}} \exp\left(-\frac{n}{2}\frac{V_\mu}{\sigma^2}\right), \tag{5.9}$$

where $V_\mu = \sum_i (x_i - \mu)^2/n$.

(c) Using (a) and (b) write a Gibbs sampler (in pseudo code) to draw a dependent sample from the posterior pdf.

3. Let $\mathbf{X} = (X, Y)^\top$ be a random column vector with a bivariate normal distribution with expectation vector $\mathbf{0} = (0, 0)^\top$ and covariance matrix

$$\Sigma = \begin{pmatrix} 1 & \varrho \\ \varrho & 1 \end{pmatrix} .$$

(a) Show that $(Y \,|\, X = x) \sim \mathsf{N}(\varrho\, x, 1 - \varrho^2)$ and $(X \,|\, Y = y) \sim \mathsf{N}(\varrho\, y, 1 - \varrho^2)$.

(b) Write a systematic Gibbs sampler to draw 10^4 samples from the bivariate distribution $\mathsf{N}(\mathbf{0}, \Sigma)$ and plot the data for $\varrho = 0, 0.7$, and 0.9.

4. Consider the 2-dimensional pdf

$$f(x_1, x_2) = c \exp(-(x_1^2 x_2^2 + x_1^2 + x_2^2 - 8x_1 - 8x_2)/2) ,$$

where c is a normalization constant.

(a) Plot the (3D) graph of f and the corresponding contour plot.

(b) Implement a Gibbs sampler to draw from f.

(c) Implement a random walk sampler to draw from f, using proposals of the form $\mathbf{Y} = \mathbf{x} + \mathbf{Z}$, where $\mathbf{Z}$ is 2-dimensional standard normal.

Chapter 6

Variance Reduction

The estimation of performance measures in Monte Carlo simulation can be made more efficient by utilizing known information about the simulation model. The more that is known about the behavior of the system, the greater the amount of variance reduction that can be achieved. The main variance reduction techniques discussed in this chapter are:

1. Antithetic random variables.
2. Control variables.
3. Conditional Monte Carlo.
4. Importance sampling.

6.1 Variance Reduction Example

Each of the variance reduction methods is illustrated using the following estimation problem concerning a bridge network. The problem is sufficiently complicated to warrant Monte Carlo simulation, while easy enough to implement, so that the workings of each technique can be concisely illustrated within the same context.

Example 6.1 (Bridge Network) Consider the undirected graph in Figure 6.1, depicting a **bridge network**.

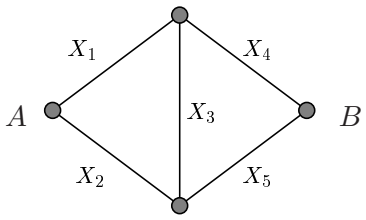


Figure 6.1: What is the expected length of the shortest path from A to B?

Suppose we wish to estimate the expected length ℓ of the shortest path between nodes (vertices) A and B, where the lengths of the links (edges) are random variables $X_1, \ldots, X_5$. We have $\ell = \mathbb{E}H(\mathbf{X})$, where

$$H(\mathbf{X}) = \min\{X_1 + X_4,\ X_1 + X_3 + X_5,\ X_2 + X_3 + X_4,\ X_2 + X_5\}\,. \tag{6.1}$$

Note that $H(\mathbf{x})$ is nondecreasing in each component of the vector $\mathbf{x}$. Suppose the lengths $\{X_i\}$ are independent and $X_i \sim \mathsf{U}(0, a_i)$, $i = 1, \ldots, 5$ with $(a_1, \ldots, a_5) = (1, 2, 3, 1, 2)$. Writing $X_i = a_i U_i$, $i = 1, \ldots, 5$ with $\{U_i\} \sim_{\text{iid}} \mathsf{U}(0, 1)$, we can restate the problem as the estimation of

$$\ell = \mathbb{E}h(\mathbf{U})\,, \tag{6.2}$$

where $\mathbf{U} = (U_1, \ldots, U_5)$ and $h(\mathbf{U}) = H(a_1 U_1, \ldots, a_5 U_5)$. The exact value can be determined by conditioning (see Section 6.4) and is given by

$$\ell = \frac{1339}{1440} = 0.9298611111\ldots\,.$$

Crude Monte Carlo (CMC) proceeds by generating $\mathbf{U}_1, \ldots, \mathbf{U}_N \overset{\text{iid}}{\sim} \mathsf{U}(0, 1)^5$ and returning

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} h(\mathbf{U}_k)$$

as an estimate for ℓ. To assess the *accuracy* of the estimate, we need to look at the probability distribution of the estimator $\widehat{\ell}$. In particular, the expectation of $\widehat{\ell}$ is ℓ (the estimator is unbiased), and the variance of $\widehat{\ell}$ is σ^2/N, where σ^2 is the variance of $h(\mathbf{U})$. Hence, if σ^2 is finite, then $\mathrm{Var}(\widehat{\ell}) \to 0$ as $N \to \infty$. Moreover, by the central limit theorem, $\widehat{\ell}$ has approximately a $\mathsf{N}(\ell, \sigma^2/N)$ distribution for large N. This implies that for large N

$$\mathbb{P}\left(\frac{|\widehat{\ell} - \ell|}{\sqrt{\sigma^2}\sqrt{N}} > z_{1-\alpha/2} \right) \approx \alpha\,,$$

where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ quantile of the standard normal distribution. Replacing σ^2 with its unbiased estimator $S^2 = \sum_{i=1}^{N} (h(\mathbf{U}_i) - \widehat{\ell})^2/(N-1)$ — the sample variance of $h(\mathbf{U}_1), \ldots, h(\mathbf{U}_N)$—, it follows that for large N

$$\mathbb{P}\left(\widehat{\ell} - z_{1-\alpha/2} \frac{S}{\sqrt{N}} \leqslant \ell \leqslant \widehat{\ell} + z_{1-\alpha/2} \frac{S}{\sqrt{N}} \right) \approx \alpha\,.$$

In other words, $(\widehat{\ell} - z_{1-\alpha/2} \frac{S}{\sqrt{N}},\ \widehat{\ell} + z_{1-\alpha/2} \frac{S}{\sqrt{N}})$ is an approximate α-**confidence interval** for ℓ. Typical values for α are 0.90 and 0.95, with corresponding (0.95 and 0.975) quantiles 1.65 and 1.96. The accuracy of an estimate $\widehat{\ell}$ is often reported by stating its estimated **relative error** $S/(\widehat{\ell}\sqrt{N})$.

The following MATLAB program implements the CMC simulation. For a sample size of $N = 10^4$ a typical estimate is $\widehat{\ell} = 0.930$ with an estimated relative error of 0.43%.

```
%bridgeCMC.m
N = 10^4;
U = rand(N,5);
y = h(U);
est = mean(y)
percRE = std(y)/sqrt(N)/est*100
```

```
function out=h(u)
a=[1,2,3,1,2]; N = size(u,1);
X = u.*repmat(a,N,1);
Path_1=X(:,1)+X(:,4);
Path_2=X(:,1)+X(:,3)+X(:,5);
Path_3=X(:,2)+X(:,3)+X(:,4);
Path_4=X(:,2)+X(:,5);
out=min([Path_1,Path_2,Path_3,Path_4],[],2);
```

6.2 Antithetic Random Variables

A pair of real-valued random variables (Y, Y^*) is called an **antithetic pair** if Y and Y^* have the same distribution and are *negatively correlated*. The main application of antithetic random variables in Monte Carlo estimation is based on the following theorem.

Theorem 6.2.1 (Antithetic Estimator) *Let N be an even number and let $(Y_1, Y_1^*), \ldots, (Y_{N/2}, Y_{N/2}^*)$ be independent antithetic pairs of random variables, where each Y_k and Y_k^* is distributed as Y. The* **antithetic estimator**

$$\widehat{\ell}^{(a)} = \frac{1}{N} \sum_{k=1}^{N/2} \{Y_k + Y_k^*\}, \tag{6.3}$$

is an unbiased estimator of $\ell = \mathbb{E}Y$, with variance

$$\begin{aligned} \mathrm{Var}(\widehat{\ell}^{(a)}) &= \frac{N/2}{N^2} \left(\mathrm{Var}(Y) + \mathrm{Var}(Y^*) + 2\,\mathrm{Cov}(Y, Y^*)\right) \\ &= \left(\mathrm{Var}(Y) + \mathrm{Cov}(Y, Y^*)\right)/N \\ &= \frac{\mathrm{Var}(Y)}{N} \left(1 + \varrho_{Y,Y^*}\right), \end{aligned}$$

where ϱ_{Y,Y^*} is the correlation between Y and Y^*.

Note that (6.3) is simply the sample mean of the independent random variables $\{(Y_k+Y_k^*)/2\}$. Since the variance of the CMC estimator $\widehat{\ell} = N^{-1}\sum_{k=1}^N Y_i$ is $\mathrm{Var}(Y)/N$, the above theorem shows that the use of antithetic variables leads to a smaller variance of the estimator by a factor of $1+\varrho_{Y,Y^*}$. The amount of reduction depends crucially on the amount of negative correlation between the antithetic variables.

In general, the output of a simulation run is of the form $Y = h(\mathbf{U})$, where h is a real-valued function and $\mathbf{U} = (U_1, U_2, \ldots)$ is a random vector of iid $\mathsf{U}(0,1)$ random variables. Suppose that $\mathbf{U}^*$ is another vector of iid $\mathsf{U}(0,1)$ random variables which is dependent on $\mathbf{U}$ and for which Y and $Y^* = h(\mathbf{U}^*)$ are negatively correlated. Then (Y, Y^*) is an antithetic pair. In particular, if h is a monotone function in each of its components, then the choice $\mathbf{U}^* = \mathbf{1} - \mathbf{U}$, where $\mathbf{1}$ is the vector of 1s, yields an antithetic pair.

An alternative to CMC for estimating $\ell = \mathbb{E}Y = \mathbb{E}h(\mathbf{U})$ is thus as follows.

Algorithm 6.1 (Antithetic Estimation for Monotone h)

1. *Generate* $Y_1 = h(\mathbf{U}_1), \ldots, Y_{N/2} = h(\mathbf{U}_{N/2})$ *from independent simulation runs.*
2. *Let* $Y_1^* = h(\mathbf{1}-\mathbf{U}_1), \ldots, Y_{N/2}^* = h(\mathbf{1}-\mathbf{U}_{N/2})$.
3. *Compute the sample covariance matrix corresponding to the pairs* $\{(Y_k, Y_k^*)\}$*:*

$$C = \begin{pmatrix} \frac{1}{N/2-1}\sum_{k=1}^{N/2}(Y_k-\bar{Y})^2 & \frac{1}{N/2-1}\sum_{k=1}^{N/2}(Y_k-\bar{Y})(Y_k^*-\bar{Y}^*) \\ \frac{1}{N/2-1}\sum_{k=1}^{N/2}(Y_k-\bar{Y})(Y_k^*-\bar{Y}^*) & \frac{1}{N/2-1}\sum_{k=1}^{N/2}(Y_k^*-\bar{Y}^*)^2 \end{pmatrix}.$$

4. *Estimate* ℓ *via the antithetic estimator* $\widehat{\ell}^{(a)}$ *in* (6.3) *and determine an approximate* $1-\alpha$ *confidence interval as*

$$\left(\widehat{\ell}^{(a)} - z_{1-\alpha/2}SE, \quad \widehat{\ell}^{(a)} + z_{1-\alpha/2}SE\right),$$

where SE is the estimated standard error:

$$SE = \sqrt{\frac{C_{1,1}+C_{2,2}+2C_{1,2}}{2N}},$$

and z_γ *denotes the* γ*-quantile of the* $\mathsf{N}(0,1)$ *distribution.*

For each of the $N/2$ runs in Step 2 one does not necessarily have to store the complete sequence $\mathbf{U} = (U_1, U_2, \ldots)$ of random numbers in memory, but simply save the random seeds for each sequence.

☞ 10

Example 6.2 (Antithetic Estimation for the Bridge Network) The following MATLAB program implements an antithetic estimator of the expected length of the shortest path ℓ in Example 6.1.

A typical estimate using $N = 10^4$ samples is $\widehat{\ell}^{(a)} = 0.929$ with an estimated relative error of 0.2%. Figure 6.2 illustrates that the correlation between $h(\mathbf{U})$

and $h(\mathbf{1}-\mathbf{U})$ is relatively high in this case. The correlation coefficient is around -0.77, which means a more than four-fold reduction in simulation effort when compared to CMC. Function `h.m` in the code that follows is the same as in Example 6.1.

```
%compare_CMC_and_ARV.m
N=10^4;
U=rand(N/2,5);  % get uniform random variables
y = h(U); ya = h(1-U);
ell=(mean(y) + mean(ya))/2;
C=cov(y,ya);
var_h = sum(sum(C))/(2*N);
corr = C(1,2)/sqrt(C(1,1)*C(2,2));
fprintf('ell= %g,RE = %g,corr = %g\n',ell,sqrt(var_h)/ell, corr)
plot(y,ya,'.')
U = rand(N,5);
yb = h(U);
var_hb = var(yb)/N;
ReB = sqrt(var_hb)/ell
```

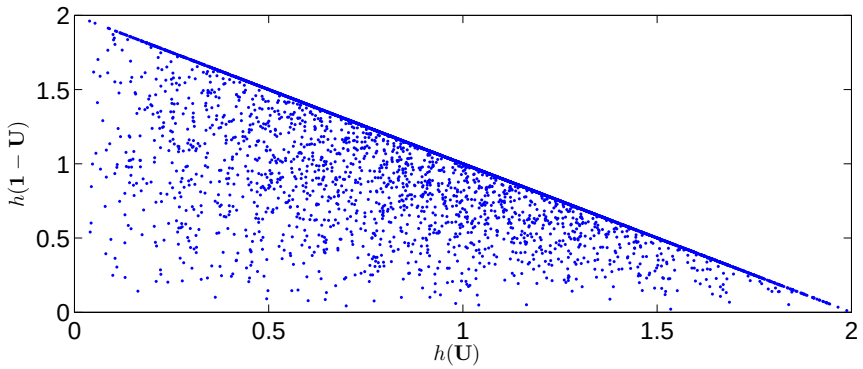


Figure 6.2: Scatter plot of $N = 10^4$ antithetic pairs (Y, Y^*) for the bridge network.

Remark 6.2.1 (Normal Antithetic Random Variables) Antithetic pairs can also be based on distributions other than the uniform. For example, suppose that $Y = H(\mathbf{Z})$, where $\mathbf{Z} = (Z_1, Z_2, \ldots)$ is a vector of iid standard normal random variables. By the inverse-transform method we can write $Y = h(\mathbf{U})$, with $h(\mathbf{u}) = H(\Phi^{-1}(u_1), \Phi^{-1}(u_2), \ldots)$, where Φ is the cdf of the $\mathsf{N}(0,1)$ distribution. Taking $\mathbf{U}^* = \mathbf{1} - \mathbf{U}$ gives $\mathbf{Z}^* = (\Phi^{-1}(U_1^*), \Phi^{-1}(U_2^*), \ldots) = -\mathbf{Z}$, so that (Y, Y^*) with $Y^* = H(-\mathbf{Z})$ forms an antithetic pair provided that Y and Y^* are negatively correlated, which is the case if H is a monotone function in each of its components.

☞ 26

6.3 Control Variables

Let Y be the output of a simulation run. A random variable $\widetilde{Y}$, obtained from the same simulation run, is called a **control variable** for Y if Y and $\widetilde{Y}$ are correlated (negatively or positively) and the expectation of $\widetilde{Y}$ is known. The use of control variables for variance reduction is based on the following observation.

Theorem 6.3.1 (Control Variable Estimation) *Let $Y_1, \ldots, Y_N$ be the output of N independent simulation runs, and let $\widetilde{Y}_1, \ldots, \widetilde{Y}_N$ be the corresponding control variables, with $\mathbb{E}\widetilde{Y}_k = \widetilde{\ell}$ known. Let $\varrho_{Y,\widetilde{Y}}$ be the correlation coefficient between each Y_k and $\widetilde{Y}_k$. For each $\alpha \in \mathbb{R}$ the (linear) estimator*

$$\widehat{\ell}^{(c)} = \frac{1}{N} \sum_{k=1}^{N} \left[Y_k - \alpha \left(\widetilde{Y}_k - \widetilde{\ell} \right) \right] \tag{6.4}$$

is an unbiased estimator for $\ell = \mathbb{E}Y$. The minimal variance of $\widehat{\ell}^{(c)}$ is

$$\mathrm{Var}(\widehat{\ell}^{(c)}) = \frac{1}{N} (1 - \varrho^2_{Y,\widetilde{Y}}) \mathrm{Var}(Y) \tag{6.5}$$

which is obtained for $\alpha = \mathrm{Cov}(Y, \widetilde{Y})/\mathrm{Var}(\widetilde{Y})$.

Usually the optimal α in (6.5) is unknown, but it can be easily estimated from the sample covariance matrix of the $\{(Y_k, \widetilde{Y}_k)\}$. This leads to the following algorithm.

Algorithm 6.2 (Control Variable Estimation)

1. *From N independent simulation runs generate $Y_1, \ldots, Y_N$ and the control variables $\widetilde{Y}_1, \ldots, \widetilde{Y}_N$.*
2. *Compute the sample covariance matrix of $\{(Y_k, \widetilde{Y}_k)\}$:*

$$C = \begin{pmatrix} \frac{1}{N-1} \sum_{k=1}^{N} (Y_k - \bar{Y})^2 & \frac{1}{N-1} \sum_{k=1}^{N} (Y_k - \bar{Y})(\widetilde{Y}_k - \bar{\widetilde{Y}}) \\ \frac{1}{N-1} \sum_{k=1}^{N} (Y_k - \bar{Y})(\widetilde{Y}_k - \bar{\widetilde{Y}}) & \frac{1}{N-1} \sum_{k=1}^{N} (\widetilde{Y}_k - \bar{\widetilde{Y}})^2 \end{pmatrix}.$$

3. *Estimate ℓ via the control variable estimator $\widehat{\ell}^{(c)}$ in (6.4) with $\alpha = C_{1,2}/C_{2,2}$ and determine an approximate $1-\alpha$ confidence interval as*

$$\left(\widehat{\ell}^{(c)} - z_{1-\alpha/2} SE, \quad \widehat{\ell}^{(c)} + z_{1-\alpha/2} SE \right),$$

where z_γ denotes the γ-quantile of the $\mathsf{N}(0,1)$ distribution and SE is the estimated standard error:

$$SE = \sqrt{\frac{1}{N} \left(1 - \frac{C_{1,2}^2}{C_{1,1} C_{2,2}} \right) C_{1,1}}.$$

Example 6.3 (Control Variable Estimation for the Bridge Network) Consider again the stochastic shortest path estimation problem for the bridge network in Example 6.1. As a control variable we can use, for example,

$$\widetilde{Y} = \min\{X_1 + X_4, X_2 + X_5\} .$$

This is particularly convenient for the current parameters $(1, 2, 3, 1, 2)$, as with high probability the shortest path will have a length equal to $\widetilde{Y}$; indeed, it will most likely have length $X_1 + X_4$, so that the latter would also be useful as a control variable. With a little calculation, the expectation of $\widetilde{Y}$ can be found to be $\mathbb{E}\widetilde{Y} = 15/16 = 0.9375$. Figure 6.3 shows the high correlation between the length of the shortest path $Y = H(\mathbf{X})$ defined in (6.1) and $\widetilde{Y}$. The corresponding correlation coefficient is around 0.98, which shows that a fifty-fold variance reduction in simulation effort is achieved compared with CMC estimation. The MATLAB program below implements the control variable estimator, using a sample size of $N = 10^4$. A typical estimate is $\widehat{\ell}^{(c)} = 0.92986$ with an estimated relative error of 0.05%. Function `h.m` in the code below is the same as in Example 6.1.

```
%bridgeCV.m
N=10^4;
u = rand(N,5);
Y = h(u);
Yc = hc(u);
plot(Y,Yc,'.')
C = cov(Y,Yc);
cor = C(1,2)/sqrt(C(1,1)*C(2,2))
alpha = C(1,2)/C(2,2);
yc = 15/16;
est = mean(Y - alpha*(Yc - yc))
RE = sqrt((1 - cor^2)*C(1,1)/N)/est
```

```
function out=hc(u)
a=[1,2,3,1,2];
N = size(u,1);
X = u.*repmat(a,N,1);
Path_1=X(:,1)+X(:,4);
Path_4=X(:,2)+X(:,5);
out=min([Path_1,Path_4],[],2);
```

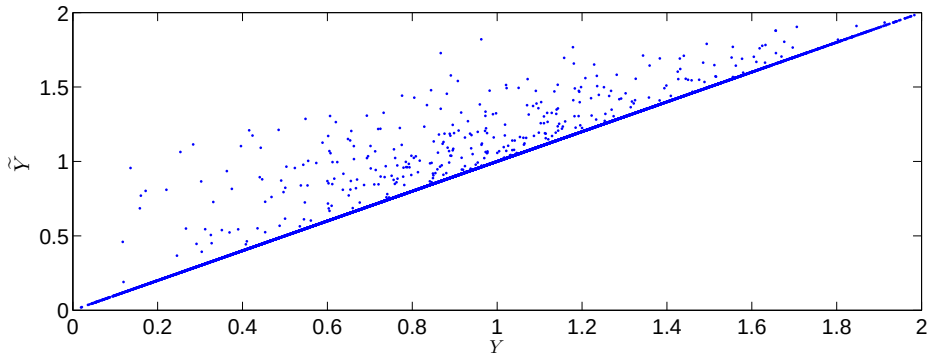


Figure 6.3: Scatter plot of $N = 10^4$ pairs $(Y, \widetilde{Y})$ for the output Y and control variable $\widetilde{Y}$ of the stochastic shortest path problem.

Remark 6.3.1 (Multiple Control Variables) Algorithm 6.2 can be extended straightforwardly to the case where more than one control variable is used for each output Y. Specifically, let $\widetilde{\mathbf{Y}} = (\widetilde{Y}_1, \ldots, \widetilde{Y}_m)^\top$ be a (column) vector of m control variables with known mean vector $\widetilde{\boldsymbol{\ell}} = \mathbb{E}\widetilde{\mathbf{Y}} = (\widetilde{\ell}_1, \ldots, \widetilde{\ell}_m)^\top$, where $\widetilde{\ell}_i = \mathbb{E}\widetilde{Y}_i$. Then, the control vector estimator of the optimal $\ell = \mathbb{E}Y$ based on independent random variables $Y_1, \ldots, Y_N$ with control vectors $\widetilde{\mathbf{Y}}_1 = (\widetilde{Y}_{11}, \ldots, \widetilde{Y}_{1m})^\top, \ldots, \widetilde{\mathbf{Y}}_N = (\widetilde{Y}_{N1}, \ldots, \widetilde{Y}_{Nm})^\top$ is given by

$$\widehat{\ell}^{(c)} = \frac{1}{N} \sum_{k=1}^{N} \left[Y_k - \boldsymbol{\alpha}^\top \left(\widetilde{\mathbf{Y}}_k - \widetilde{\boldsymbol{\ell}} \right) \right],$$

where $\boldsymbol{\alpha}$ is an estimator of the optimal vector $\boldsymbol{\alpha}^* = \Sigma_{\widetilde{\mathbf{Y}}}^{-1} \boldsymbol{\sigma}_{Y,\widetilde{\mathbf{Y}}}$. Here $\Sigma_{\widetilde{\mathbf{Y}}}$ is the $m \times m$ covariance matrix of $\widetilde{\mathbf{Y}}$, and $\boldsymbol{\sigma}_{Y,\widetilde{\mathbf{Y}}}$ is the $m \times 1$ vector whose i-th component is the covariance of Y and $\widetilde{Y}_i$, $i = 1, \ldots, m$. The variance of $\widehat{\ell}^{(c)}$ for $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$ is

$$\mathrm{Var}(\widehat{\ell}^{(c)}) = \frac{1}{N}(1 - R^2_{Y,\widetilde{\mathbf{Y}}})\mathrm{Var}(Y) \,, \tag{6.6}$$

where

$$R^2_{Y,\widetilde{\mathbf{Y}}} = (\boldsymbol{\sigma}_{Y,\widetilde{\mathbf{Y}}})^\top \Sigma_{\widetilde{\mathbf{Y}}}^{-1} \boldsymbol{\sigma}_{Y,\widetilde{\mathbf{Y}}} / \mathrm{Var}(Y)$$

is the square of the **multiple correlation coefficient** of Y and $\widetilde{\mathbf{Y}}$. Again, the larger $R^2_{Y,\widetilde{\mathbf{Y}}}$ is, the greater the variance reduction.

6.4 Conditional Monte Carlo

Variance reduction using **conditional Monte Carlo** is based on the following result.

Theorem 6.4.1 (Conditional Variance) *Let Y be a random variable and $\mathbf{Z}$ a random vector. Then*

$$\mathrm{Var}(Y) = \mathbb{E}\,\mathrm{Var}(Y \,|\, \mathbf{Z}) + \mathrm{Var}(\,\mathbb{E}[Y \,|\, \mathbf{Z}]\,) \,, \tag{6.7}$$

and hence $\mathrm{Var}(\mathbb{E}[Y \mid \mathbf{Z}]) \leqslant \mathrm{Var}(Y)$.

Suppose that the aim is to estimate $\ell = \mathbb{E}Y$, where Y is the output from a Monte Carlo experiment, and that one can find a random variable (or vector), $\mathbf{Z} \sim g$, such that the conditional expectation $\mathbb{E}[Y \mid \mathbf{Z} = \mathbf{z}]$ can be computed analytically. By the tower property,

$$\ell = \mathbb{E}Y = \mathbb{E}\,\mathbb{E}[Y \mid \mathbf{Z}] \;, \tag{6.8}$$

it follows that $\mathbb{E}[Y \mid \mathbf{Z}]$ is an unbiased estimator of ℓ. Moreover, by Theorem 6.4.1 the variance of $\mathbb{E}[Y \mid \mathbf{Z}]$ is always smaller than or equal to the variance of Y. The conditional Monte Carlo idea is sometimes referred to as **Rao–Blackwellization**.

Algorithm 6.3 (Conditional Monte Carlo)

1. *Generate a sample* $\mathbf{Z}_1, \ldots, \mathbf{Z}_N \overset{\text{iid}}{\sim} g$.
2. *Calculate* $\mathbb{E}[Y \mid \mathbf{Z}_k]$, $k = 1, \ldots, N$ *analytically.*
3. *Estimate* $\ell = \mathbb{E}Y$ *by*

$$\widehat{\ell}_c = \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}[Y \mid \mathbf{Z}_k] \tag{6.9}$$

and determine an approximate $1-\alpha$ *confidence interval as*

$$\left(\widehat{\ell}_c - z_{1-\alpha/2} \frac{S}{\sqrt{N}}, \quad \widehat{\ell}_c + z_{1-\alpha/2} \frac{S}{\sqrt{N}} \right) \;,$$

where S *is the sample standard deviation of the* $\{\mathbb{E}[Y \mid \mathbf{Z}_k]\}$ *and* z_γ *denotes the* γ*-quantile of the* $\mathsf{N}(0,1)$ *distribution.*

Example 6.4 (Conditional Monte Carlo for the Bridge Network) We return to Example 6.1. Let $Z_1 = \min\{X_4, X_3 + X_5\}$, $Z_2 = \min\{X_5, X_3 + X_4\}$, $Y_1 = X_1 + Z_1$, $Y_2 = X_2 + Z_2$, and $\mathbf{Z} = (Z_1, Z_2)$. Then, $Y = H(\mathbf{X})$ can be written as

$$Y = \min\{Y_1, Y_2\} \;,$$

where conditional upon $\mathbf{Z} = \mathbf{z}$, (Y_1, Y_2) is uniformly distributed on the rectangle $\mathscr{R}_{\mathbf{z}} = [z_1, z_1 + 1] \times [z_2, z_2 + 2]$. The conditional expectation of Y given $\mathbf{Z} = \mathbf{z}$ can be evaluated exactly, and is given by

$$\mathbb{E}[Y \mid \mathbf{Z} = \mathbf{z}] = \begin{cases} \frac{1}{2} + z_1 & \text{if } \mathbf{z} \in \mathscr{A}_0 \;, \\ \frac{5}{12} + \frac{3z_1}{4} - \frac{z_1^2}{4} - \frac{z_1^3}{12} + \frac{z_2}{4} + \frac{z_1 z_2}{2} + \frac{z_1^2 z_2}{4} - \frac{z_2^2}{4} - \frac{z_1 z_2^2}{4} + \frac{z_2^3}{12} & \text{if } \mathbf{z} \in \mathscr{A}_1 \;, \\ \frac{1}{12}(5 - 3z_1^2 + 3z_2 - 3z_2^2 + z_1(9 + 6z_2)) & \text{if } \mathbf{z} \in \mathscr{A}_2 \;, \end{cases}$$

where

$$\begin{aligned} \mathscr{A}_0 &= \{\mathbf{z} : 0 \leqslant z_1 \leqslant 1,\ z_1 + 1 \leqslant z_2 \leqslant 2\}, \\ \mathscr{A}_1 &= \{\mathbf{z} : 0 \leqslant z_1 \leqslant 1,\ z_1 \leqslant z_2 \leqslant z_1 + 1\}, \\ \mathscr{A}_2 &= \{\mathbf{z} : 0 \leqslant z_1 \leqslant 1,\ 0 \leqslant z_2 \leqslant z_1\} \;. \end{aligned}$$

For example, if $\mathbf{z} \in \mathscr{A}_1$, then the domain $\mathscr{R}_{\mathbf{z}}$ of (Y_1, Y_2) intersects the line $y_1 = y_2$ at $y_1 = z_2$ and $y_1 = z_1 + 1$, so that

$$\mathbb{E}[Y \,|\, \mathbf{Z} = \mathbf{z}] = \int_{z_1}^{z_2} \int_{z_2}^{z_2+2} y_1 \, \frac{1}{2} \, \mathrm{d}y_2 \, \mathrm{d}y_1 + \int_{z_2}^{z_1+1} \int_{y_1}^{z_2+2} y_1 \, \frac{1}{2} \, \mathrm{d}y_2 \, \mathrm{d}y_1 + \int_{z_2}^{z_1+1} \int_{z_2}^{y_1} y_2 \, \frac{1}{2} \, \mathrm{d}y_2 \, \mathrm{d}y_1 \, .$$

The following MATLAB program gives an implementation of the corresponding conditional Monte Carlo estimator. A typical outcome for sample size $N = 10^4$ is $\widehat{\ell}_c = 0.9282$ with an estimated relative error of 0.29%, compared with 0.43% for CMC, indicating more than a two-fold reduction in simulation effort. Interestingly, the joint pdf of $\mathbf{Z}$ on $[0,1] \times [0,2]$ can, with considerable effort, be determined analytically, so that $\ell = \mathbb{E}Y$ can be evaluated exactly. This leads to the exact value given in the introduction:

$$\mathbb{E}Y = \frac{1339}{1440} = 0.9298611111\ldots \, .$$

```
%bridgeCondMC.m
N = 10^4;
S = zeros(N,1);
for i = 1:N
    u = rand(1,5);
    Z = Zcond(u);
    if Z(2)> Z(1) + 1
        S(i) = 1/2 + Z(1);
    elseif (Z(2) > Z(1))
        S(i) = 5/12 + (3*Z(1))/4 - Z(1)^2/4 - Z(1)^3/12 ...
             + Z(2)/4 + (Z(1)*Z(2))/2 + (Z(1)^2*Z(2))/4 ...
             -  Z(2)^2/4 - (Z(1)*Z(2)^2)/4 + Z(2)^3/12;
    else
        S(i) = (5 - 3*Z(1)^2 + 3*Z(2) - 3*Z(2)^2  ...
             + Z(1)*(9 + 6*Z(2)))/12;
    end
end
est = mean(S)
RE = std(S)/sqrt(N)/est
```

```
function Z=Zcond(u)
a=[1,2,3,1,2];
X = u*diag(a);
Z = [min([X(:,4), X(:,3) + X(:,5)],[],2),...
    min([X(:,5), X(:,3) + X(:,4)],[],2)];
```

6.5 Importance Sampling

One of the most important variance reduction techniques is **importance sampling**. This technique is especially useful for the estimation of rare-event probabilities. The standard setting is the estimation of a quantity

$$\ell = \mathbb{E}_f H(\mathbf{X}) = \int H(\mathbf{x})\, f(\mathbf{x})\, \mathrm{d}\mathbf{x}\,, \tag{6.10}$$

where H is a real-valued function and f the probability density of a random vector $\mathbf{X}$, called the **nominal pdf**. The subscript f is added to the expectation operator to indicate that it is taken with respect to the density f.

Let g be another probability density such that $H\,f$ is **dominated** by g. That is, $g(\mathbf{x}) = 0 \;\Rightarrow\; H(\mathbf{x})\, f(\mathbf{x}) = 0$. Using the density g we can represent ℓ as

$$\ell = \int H(\mathbf{x})\, \frac{f(\mathbf{x})}{g(\mathbf{x})}\, g(\mathbf{x})\, \mathrm{d}\mathbf{x} = \mathbb{E}_g H(\mathbf{X})\, \frac{f(\mathbf{X})}{g(\mathbf{X})}\,. \tag{6.11}$$

Consequently, if $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} g$, then

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \frac{f(\mathbf{X}_k)}{g(\mathbf{X}_k)} \tag{6.12}$$

is an unbiased estimator of ℓ. This estimator is called the **importance sampling estimator** and g is called the importance sampling density. The ratio of densities,

$$W(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}\,, \tag{6.13}$$

is called the **likelihood ratio** — with a slight abuse of nomenclature, as the likelihood is usually seen in statistics as a function of the parameters.

Algorithm 6.4 (Importance Sampling Estimation)

1. *Select an importance sampling density g that dominates Hf.*
2. *Generate $\mathbf{X}_1, \ldots, \mathbf{X}_N \overset{\text{iid}}{\sim} g$ and let $Y_i = H(\mathbf{X}_i) f(\mathbf{X}_i)/g(\mathbf{X}_i)$, $i = 1, \ldots, N$.*
3. *Estimate ℓ via $\widehat{\ell} = \bar{Y}$ and determine an approximate $1-\alpha$ confidence interval as*
$$\left(\widehat{\ell} - z_{1-\alpha/2} \frac{S}{\sqrt{N}},\; \widehat{\ell} + z_{1-\alpha/2} \frac{S}{\sqrt{N}} \right),$$
where z_γ denotes the γ-quantile of the $\mathsf{N}(0,1)$ distribution and S is the sample standard deviation of $Y_1, \ldots, Y_N$.

Example 6.5 (Importance Sampling for the Bridge Network) The expected length of the shortest path in Example 6.1 can be written as (see (6.2))

$$\ell = \mathbb{E}h(\mathbf{U}) = \int h(\mathbf{u})\, \mathrm{d}\mathbf{u}\,,$$

where $\mathbf{U} = (U_1, \ldots, U_5)$ and $U_1, \ldots, U_5 \overset{\text{iid}}{\sim} \mathsf{U}(0,1)$. The nominal pdf is thus $f(\mathbf{u}) = 1, \mathbf{u} \in (0,1)^5$. Suppose the importance sampling pdf is of the form

$$g(\mathbf{u}) = \prod_{i=1}^{5} \nu_i \, u_i^{\nu_i - 1} \,,$$

which means that under g the components of $\mathbf{U}$ are again independent and $U_i \sim \mathsf{Beta}(\nu_i, 1)$ for some $\nu_i > 0$, $i = 1, \ldots, 5$. For the nominal (uniform) distribution we have $\nu_i = 1$, $i = 1, \ldots, 5$. Generating $\mathbf{U}$ under g is easily carried out via the inverse-transform method. A good choice of $\{\nu_i\}$ is of course crucial. The MATLAB program below implements the importance sampling estimation of ℓ using, for example, $(\nu_1, \ldots, \nu_5) = (1.3, 1.1, 1, 1.3, 1.1)$. For a sample size of $N = 10^4$ a typical estimate is $\widehat{\ell} = 0.9295$ with an estimated relative error of 0.24%, which gives about a four-fold reduction in simulation effort compared with CMC estimation, despite the fact that the $\{\nu_i\}$ are all quite close to their nominal value 1.

```
%bridgeIS.m
N = 10^4;
nu0 = [1.3, 1.1, 1, 1.3, 1.1];
nu = repmat(nu0,N,1);
U = rand(N,5).^(1./nu);
W = prod(1./(nu.*U.^(nu - 1)),2);
y = h(U).*W;
est = mean(y)
percRE = std(y)/sqrt(N)/est*100
```

The main difficulty in importance sampling is how to choose the importance sampling distribution. A poor choice of g may seriously compromise the estimate and the confidence intervals. The following sections provide some guidance toward choosing a good importance sampling distribution.

6.5.1 Minimum-Variance Density

The optimal choice g^* for the importance sampling density minimizes the variance of $\widehat{\ell}$, and is therefore the solution to the functional minimization program

$$\min_g \mathrm{Var}_g \left(H(\mathbf{X}) \, \frac{f(\mathbf{X})}{g(\mathbf{X})} \right) \,. \tag{6.14}$$

It is not difficult to show that

$$g^*(\mathbf{x}) = \frac{|H(\mathbf{x})| \, f(\mathbf{x})}{\int |H(\mathbf{x})| \, f(\mathbf{x}) \, \mathrm{d}\mathbf{x}} \,. \tag{6.15}$$

In particular, if $H(\mathbf{x}) \geqslant 0$ or $H(\mathbf{x}) \leqslant 0$ then

$$g^*(\mathbf{x}) = \frac{H(\mathbf{x}) \, f(\mathbf{x})}{\ell} \,, \tag{6.16}$$

in which case $\mathrm{Var}_{g^*}(\widehat{\ell}) = \mathrm{Var}_{g^*}(H(\mathbf{X})W(\mathbf{X})) = \mathrm{Var}_{g^*}(\ell) = 0$, so that the estimator $\widehat{\ell}$ is *constant* under g^*. An obvious difficulty is that the evaluation of the optimal importance sampling density g^* is usually not possible. For example, $g^*(\mathbf{x})$ in (6.16) depends on the unknown quantity ℓ. Nevertheless, a good importance sampling density g should be "close" to the minimum variance density g^*.

One of the main considerations for choosing a good importance sampling pdf is that the estimator (6.12) should have finite variance. This is equivalent to the requirement that

$$\mathbb{E}_g H^2(\mathbf{X}) \frac{f^2(\mathbf{X})}{g^2(\mathbf{X})} = \mathbb{E}_f H^2(\mathbf{X}) \frac{f(\mathbf{X})}{g(\mathbf{X})} < \infty \,. \tag{6.17}$$

This suggests that g should not have lighter tails than f, and that, preferably, the likelihood ratio, f/g, should be bounded.

6.5.2 Variance Minimization Method

When the pdf f belongs to some parametric family of distributions, it is often convenient to choose the importance sampling distribution from the *same* family. In particular, suppose that $f(\cdot;\boldsymbol{\theta})$ belongs to the family

$$\{f(\cdot;\boldsymbol{\eta}),\ \boldsymbol{\eta} \in \Theta\} \,.$$

Then, the problem of finding an optimal importance sampling density in this class reduces to the following *parametric* minimization problem

$$\min_{\boldsymbol{\eta}\in\Theta} \mathrm{Var}_{\boldsymbol{\eta}} \left(H(\mathbf{X})\, W(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\eta})\right) \,, \tag{6.18}$$

where $W(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\eta}) = f(\mathbf{X};\boldsymbol{\theta})/f(\mathbf{X};\boldsymbol{\eta})$. We call $\boldsymbol{\theta}$ the **nominal parameter** and $\boldsymbol{\eta}$ the **reference parameter vector** or **tilting vector**. Since under any $f(\cdot;\boldsymbol{\eta})$ the expectation of $H(\mathbf{X})\, W(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\eta})$ is ℓ, the optimal solution of (6.18) coincides with that of

$$\min_{\boldsymbol{\eta}\in\Theta} V(\boldsymbol{\eta}) \,, \tag{6.19}$$

where

$$V(\boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{\eta}} H^2(\mathbf{X})\, W^2(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{\theta}} H^2(\mathbf{X})\, W(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\eta}) \,. \tag{6.20}$$

We call either of the equivalent problems (6.18) and (6.19) the **variance minimization** (VM) problem; and we call the parameter vector ${}_*\boldsymbol{\eta}$ that minimizes the programs (6.18) – (6.19) the **VM-optimal reference parameter vector**. The VM problem can be viewed as a stochastic optimization problem, and can be approximately solved via Monte Carlo simulation by considering the sample average version of (6.19) – (6.20): ☞ 142

$$\min_{\boldsymbol{\eta}\in\Theta} \widehat{V}(\boldsymbol{\eta}) \,, \tag{6.21}$$

where

$$\widehat{V}(\boldsymbol{\eta}) = \frac{1}{N}\sum_{k=1}^{N} H^2(\mathbf{X}_k)\, W(\mathbf{X}_k;\boldsymbol{\theta},\boldsymbol{\eta})\,, \tag{6.22}$$

and $\mathbf{X}_1,\ldots,\mathbf{X}_N \sim_{\text{iid}} f(\cdot;\boldsymbol{\theta})$. This problem can be solved via standard numerical optimization techniques. This gives the following modification of Algorithm 6.4.

Algorithm 6.5 (Variance Minimization Method)

1. *Select a parameterized family of importance sampling densities* $\{f(\cdot;\boldsymbol{\eta})\}$.
2. *Generate a pilot sample* $\mathbf{X}_1,\ldots,\mathbf{X}_N \overset{\text{iid}}{\sim} f(\cdot;\boldsymbol{\theta})$*, and determine the solution* ${}_*\widehat{\boldsymbol{\eta}}$ *to the variance minimization problem* (6.21).
3. *Generate* $\mathbf{X}_1,\ldots,\mathbf{X}_{N_1} \overset{\text{iid}}{\sim} f(\cdot;\,{}_*\widehat{\boldsymbol{\eta}})$ *and let* $Y_i = H(\mathbf{X}_i) f(\mathbf{X}_i;\boldsymbol{\theta}) / f(\mathbf{X}_i;\,{}_*\widehat{\boldsymbol{\eta}})$, $i = 1,\ldots,N_1$.
4. *Estimate* ℓ *via* $\widehat{\ell} = \bar{Y}$ *and determine an approximate* $1-\alpha$ *confidence interval as*
$$\left(\widehat{\ell} - z_{1-\alpha/2}\frac{S}{\sqrt{N_1}},\ \widehat{\ell} + z_{1-\alpha/2}\frac{S}{\sqrt{N_1}}\right),$$
where z_γ *denotes the* γ*-quantile of the* $\mathsf{N}(0,1)$ *distribution and* S *is the sample standard deviation of* $Y_1,\ldots,Y_{N_1}$.

Example 6.6 (Variance Minimization for the Bridge Network) Consider the importance sampling approach for the bridge network in Example 6.5. There, the importance sampling distribution is the joint distribution of independent $\mathsf{Beta}(\nu_i, 1)$ random variables, for $i = 1,\ldots,5$. Hence, the reference parameter is $\boldsymbol{\nu} = (\nu_1,\ldots,\nu_5)$.

The following MATLAB program determines the optimal reference parameter vector ${}_*\widehat{\boldsymbol{\nu}}$ via the VM method using a pilot run of size $N = 10^3$ and the standard MATLAB minimization routine `fminsearch`. A typical value for ${}_*\widehat{\boldsymbol{\nu}}$ is $(1.262, 1.083,\ 1.016, 1.238, 1.067)$, which is similar to the one used in Example 6.5; the relative error is thus around 0.24%.

```
%vmceopt.m
N = 10^3;
U = rand(N,5);
[nu0,minv] =fminsearch(@(nu)f_var(nu,U,N),ones(1,5))
N1 = 10^4;
nu = repmat(nu0,N1,1);
U = rand(N1,5).^(1./nu);
w = prod(1./(nu.*U.^(nu - 1)),2);
y = h(U).*w;
est = mean(y)
percRE = std(y)/sqrt(N1)/est*100
```

```
function out = f_var(nu,U,N)
nu1 = repmat(nu,N,1);
W = prod(1./(nu1.*U.^(nu1 - 1)),2);
y = H(U);
out = W'*y.^2;
```

6.5.3 Cross-Entropy Method

An alternative approach to the VM method for choosing an "optimal" importance sampling distribution is based on the Kullback–Leibler cross-entropy distance, or simply **cross-entropy** (CE) distance. The CE distance between two continuous pdfs g and h is given by

$$\begin{aligned}\mathcal{D}(g,h) &= \mathbb{E}_g \ln \frac{g(\mathbf{X})}{h(\mathbf{X})} = \int g(\mathbf{x}) \ln \frac{g(\mathbf{x})}{h(\mathbf{x})}\,\mathrm{d}\mathbf{x} \\ &= \int g(\mathbf{x}) \ln g(\mathbf{x})\,\mathrm{d}\mathbf{x} - \int g(\mathbf{x}) \ln h(\mathbf{x})\,\mathrm{d}\mathbf{x}\,.\end{aligned} \tag{6.23}$$

For discrete pdfs replace the integrals with the corresponding sums. Observe that, by Jensen's inequality, $\mathcal{D}(g,h) \geqslant 0$, with equality if and only if $g = h$. The CE distance is sometimes called the Kullback–Leibler *divergence*, because it is not symmetric, that is, $\mathcal{D}(g,h) \neq \mathcal{D}(h,g)$ for $g \not\equiv h$.

The idea of the CE method is to choose the importance sampling density, h say, such that the CE distance between the optimal importance sampling density g^* in (6.15) and h, is minimal. We call this the **CE-optimal pdf**. This pdf solves the *functional* optimization program $\min_h \mathcal{D}(g^*, h)$. If we optimize over all densities h, then it is immediate that the CE-optimal pdf coincides with the VM-optimal pdf g^*.

As with the VM approach in (6.18) and (6.19), we shall restrict ourselves to a parametric family of densities $\{f(\cdot;\boldsymbol{\eta}), \boldsymbol{\eta} \in \Theta\}$ that contains the nominal density $f(\cdot;\boldsymbol{\theta})$. Moreover, without any loss of generality, we only consider positive functions H. The CE method now aims to solve the *parametric* optimization problem

$$\min_{\boldsymbol{\eta}\in\Theta} \mathcal{D}\left(g^*, f(\cdot;\boldsymbol{\eta})\right)\,. \tag{6.24}$$

The optimal solution coincides with that of

$$\max_{\boldsymbol{\eta}\in\Theta} D(\boldsymbol{\eta})\,, \tag{6.25}$$

where

$$D(\boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{\theta}} H(\mathbf{X}) \ln f(\mathbf{X};\boldsymbol{\eta})\,. \tag{6.26}$$

Similar to the VM program (6.19), we call either of the equivalent programs (6.24) and (6.25) the **CE program**; and we call the parameter vector $\boldsymbol{\eta}^*$ that minimizes the program (6.24) and (6.25) the **CE-optimal reference parameter**.

☞ 142

Similar to (6.21) we can estimate $\boldsymbol{\eta}^*$ via the stochastic counterpart method as the solution of the stochastic program

$$\max_{\boldsymbol{\eta}} \widehat{D}(\boldsymbol{\eta}) = \max_{\boldsymbol{\eta}} \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k) \ln f(\mathbf{X}_k; \boldsymbol{\eta}) \,, \tag{6.27}$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} f(\cdot; \boldsymbol{\theta})$.

In typical applications the function $\widehat{D}$ in (6.27) is convex and differentiable with respect to $\boldsymbol{\eta}$. In such cases the solution of (6.27) may be obtained by solving (with respect to $\boldsymbol{\eta}$) the following system of equations:

$$\frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k) \nabla \ln f(\mathbf{X}_k; \boldsymbol{\eta}) = \mathbf{0} \,, \tag{6.28}$$

where the gradient is with respect to $\boldsymbol{\eta}$. Various numerical and theoretical studies have shown that the solutions to the VM and CE programs are qualitatively similar. The main advantage of the CE approach over the VM approach is that the solution to (6.27) (or (6.28)) can often be found *analytically*, as specified in the following theorem.

Theorem 6.5.1 (Exponential Families) *If the importance sampling density is of the form*

$$f(\mathbf{x}; \boldsymbol{\eta}) = \prod_{i=1}^{n} f_i(x_i; \eta_i) \,,$$

where each $\{f_i(x_i; \eta_i), \eta_i \in \Theta_i\}$ *forms a 1-parameter exponential family parameterized by the mean, then the solution to the CE program* (6.27) *is* $\widehat{\boldsymbol{\eta}}^* = (\widehat{\eta}_1^*, \ldots, \widehat{\eta}_n^*)$*, with*

$$\widehat{\eta}_i^* = \frac{\sum_{k=1}^{N} H(\mathbf{X}_k) X_{ki}}{\sum_{k=1}^{N} H(\mathbf{X}_k)} \,, \quad i = 1, \ldots, n \,, \tag{6.29}$$

where X_{ki} *is the* i*-th coordinate of* $\mathbf{X}_k$*.*

For rare-event simulation the random variable $H(\mathbf{X})$ often takes the form of an indicator $\mathrm{I}_{\{S(\mathbf{X}) \geqslant \gamma\}}$. If the event $\{S(\mathbf{X}) \geqslant \gamma\}$ is rare under $f(\cdot; \boldsymbol{\theta})$, then with high probability the numerator and denominator in (6.29) are both zero, so that the CE-optimal parameter cannot be estimated in this way. Chapter 9 discusses how this can be remedied by using a multilevel approach or by sampling directly from the zero-variance importance sampling pdf g^*.

Example 6.7 (CE Method for the Bridge Network) In Example 6.6 the VM-optimal reference parameter is obtained by numerical minimization. We can use the CE method instead by applying (6.29) after suitable reparameterization. Note that for each i, $\mathsf{Beta}(\nu_i, 1)$ forms an exponential family, and that the corresponding expectation is $\eta_i = \nu_i/(1+\nu_i)$. It follows that the assignment $\nu_i = \eta_i/(1-\eta_i)$ reparameterizes the family in terms of the mean η_i.

The first four lines of the following MATLAB program implement the CE method for estimating the CE-optimal reference parameter. A typical outcome is $\widehat{\boldsymbol{\eta}} = (0.560, 0.529, 0.500, 0.571, 0.518)$, so that $\widehat{\boldsymbol{\nu}} = (1.272, 1.122, 1.000, 1.329, 1.075)$, which gives comparable results to the VM-optimal parameter vector. The corresponding relative error is estimated as 0.25%.

```
%bridgeCE.m
N = 10^3;
U = rand(N,5);
y = repmat(h(U),1,5);
v = sum(y.*U)./sum(y)
N1 = 10^4;
nu = repmat(v./(1-v),N1,1);
U = rand(N1,5).^(1./nu);
w = prod(1./(nu.*U.^(nu - 1)),2);
y = h(U).*w;
est = mean(y)
percRE = std(y)/sqrt(N1)/est*100
```

6.6 Exercises

1. Estimate

$$\ell = \int_0^1 \int_0^1 \frac{\sin(x)\,\mathrm{e}^{-(x+y)}}{\ln(1+x)}\,\mathrm{d}x\,\mathrm{d}y$$

via crude Monte Carlo, and give a 95% confidence interval.

2.We wish to estimate $\ell = \int_{-2}^{2} \mathrm{e}^{-x^2/2}\,\mathrm{d}x = \int H(x) f(x)\,\mathrm{d}x$ via Monte Carlo simulation using two different approaches: (1) defining $H(x) = 4\,\mathrm{e}^{-x^2/2}$ and f the pdf of the $\mathsf{U}[-2,2]$ distribution and (2) defining $H(x) = \sqrt{2\pi}\, I_{\{-2 \leqslant x \leqslant 2\}}$ and f the pdf of the $\mathsf{N}(0,1)$ distribution.

(a) For both cases estimate ℓ via the estimator $\widehat{\ell}$

$$\widehat{\ell} = N^{-1} \sum_{i=1}^{N} H(\mathbf{X}_i)\ . \tag{6.30}$$

Use a sample size of $N = 1000$.

(b) For both cases estimate the relative error κ of $\widehat{\ell}$.

(c) Give a 95% confidence interval for ℓ for both cases.

(d) Using the estimate found in (b), assess how large N should be such that the relative width of the confidence interval is less than 0.01, and carry out the simulation with this N. Compare the result with the true (numerical) value of ℓ.

3. Let $H(\mathbf{x}) = x_2/(1+|x_1|)$ and

$$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim \mathsf{N}\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 & 3 \\ 3 & 9 \end{pmatrix} \right) .$$

Estimate $\ell = H(\mathbf{X})$

(a) via crude Monte Carlo,

(b) via a control variable X_2,

(c) via conditional Monte Carlo, by conditioning on X_1.

Which is the most accurate method in terms of relative error?

4. Let $Z \sim \mathsf{N}(0,1)$. Estimate $\mathbb{P}(Z > 4)$ via importance sampling, using a shifted exponential sampling pdf:

$$g(x) = \lambda \mathrm{e}^{-\lambda(x-4)}, \quad x \geqslant 4$$

for some λ. Find a good λ, e.g., via the cross-entropy method, and choose N large enough to obtain accuracy to at least three significant digits. Compare with the exact value.

5. Consider the estimation of the tail probability $\ell = \mathbb{P}(X \geqslant \gamma)$ of some random variable X where γ is large. The crude Monte Carlo (CMC) estimator of ℓ is

$$\widehat{\ell} = \frac{1}{N} \sum_{i=1}^{N} Z_i \tag{6.31}$$

where $X_1, \ldots, X_N$ is are iid copies of X and $Z_i = \mathrm{I}_{\{X_i \geqslant \gamma\}}$, $i = 1, \ldots, N$.

(a) Express the relative error of $\widehat{\ell}$ in terms of N and ℓ.

(b) We say that an estimator (6.31) of $\ell = \mathbb{E}Z$ is *logarithmically efficient* if

$$\lim_{\gamma \to \infty} \frac{\ln \mathbb{E} Z^2}{\ln \ell^2} = 1 . \tag{6.32}$$

Show that the CMC estimator is not logarithmically efficient.

6. Prove (see (6.14)) that the solution of

$$\min_g \mathrm{Var}_g \left(H(\mathbf{X}) \frac{f(\mathbf{X})}{g(\mathbf{X})} \right)$$

is

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

7. Let $\ell = \mathbb{E}_{\boldsymbol{\theta}} H(\mathbf{X})$, with $\mathbf{X}$ distributed according to $f(\mathbf{x}; \boldsymbol{\theta})$, from the exponential family

$$f(\mathbf{x}; \boldsymbol{\eta}) = c(\boldsymbol{\eta}) \, \mathrm{e}^{\boldsymbol{\eta} \cdot \mathbf{t}(\mathbf{x})} \, h(\mathbf{x}) , \tag{6.33}$$

with $\mathbf{t}(\mathbf{x}) = (t_1(\mathbf{x}), \ldots, t_m(\mathbf{x}))^T$ and $\boldsymbol{\eta} \cdot \mathbf{t}(\mathbf{x})$ denoting the inner product $\sum_{i=1}^m \eta_i t_i(\mathbf{x})$. Suppose we wish to estimate ℓ via importance sampling using a pdf $f(\mathbf{x}; \boldsymbol{\eta})$ for some $\boldsymbol{\eta}$.

(a) Show that the CE-optimal parameter $\boldsymbol{\eta}$ satisfies

$$\mathbb{E}_{\boldsymbol{\theta}}\left[H(\mathbf{X})\left(\frac{\nabla c(\boldsymbol{\eta})}{c(\boldsymbol{\eta})}+\mathbf{t}(\mathbf{X})\right)\right]=\mathbf{0}\,. \tag{6.34}$$

(b) As an application of (6.34), suppose that we wish to estimate the expectation of $H(X)$, with $X \sim \mathsf{Exp}(\lambda_0)$. Show that the corresponding CE optimal parameter is

$$\lambda^* = \frac{\mathbb{E}_{\lambda_0} H(X)}{\mathbb{E}_{\lambda_0}[H(X)X]}\,.$$

(c) Explain how to estimate λ^* via simulation.

Chapter 7

Estimation of Derivatives

In this chapter we discuss three methods for gradient estimation: the *finite difference method*, *infinitesimal perturbation analysis*, and the *likelihood ratio* or *score function method*. The efficient estimation of derivatives is important in sensitivity analysis of simulation output and in stochastic or noisy optimization. Details on noisy optimization are given in Chapter 8. ☞ 137

7.1 Gradient Estimation

It is often the case that the performance measure ℓ from a Monte Carlo simulation can be viewed as a function of various parameters used in the simulation. These parameters can pertain to the distributions used in the simulation and to the mechanism under which the simulation is carried out. A typical setting is where ℓ is the expected output of a random variable Y whose value is dependent on a simulation parameter vector $\boldsymbol{\theta}$, such that

$$\ell(\boldsymbol{\theta}) = \mathbb{E}Y = \mathbb{E}_{\boldsymbol{\theta}_2} H(\mathbf{X}; \boldsymbol{\theta}_1) = \int H(\mathbf{x}; \boldsymbol{\theta}_1) f(\mathbf{x}; \boldsymbol{\theta}_2)\, \mathrm{d}\mathbf{x}\,, \tag{7.1}$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$, $H(\cdot; \boldsymbol{\theta}_1)$ is a sample performance function, and $f(\cdot; \boldsymbol{\theta}_2)$ is a pdf (for the discrete case replace the integral with a sum). In this context we refer to the parameter $\boldsymbol{\theta}_1$ as a **structural** parameter and $\boldsymbol{\theta}_2$ as a **distributional** parameter. An estimation problem formulated with only distributional parameters can often be transformed into one with only structural parameters, and vice versa; see Remark 7.1.1. It should be noted, however, that not all performance measures are of the form (7.1). For example, $\ell(\boldsymbol{\theta})$ could be the median or a quantile of $Y \sim f(\mathbf{y}; \boldsymbol{\theta})$.

In addition to estimating $\ell(\boldsymbol{\theta})$ it is often relevant to estimate various *derivatives* (gradients, Hessians, etc.) of $\ell(\boldsymbol{\theta})$. Two main applications are:

1. *Sensitivity analysis*: The gradient $\nabla\ell(\boldsymbol{\theta})$ indicates how sensitive the output $\ell(\boldsymbol{\theta})$ is to small changes in the input parameters $\boldsymbol{\theta}$, and can thus be used to identify its most significant components.

2. *Stochastic optimization and root finding*: Gradient estimation is closely related to optimization through the root-finding problem $\nabla\ell(\boldsymbol{\theta}) = \mathbf{0}$, as

any solution of the latter is a stationary point of ℓ and hence a candidate for a local maximum or minimum. Estimating the gradient via simulation can therefore be used to approximately determine the optimal solution(s), leading to gradient-based *noisy optimization* algorithms; see Sections 8.1–8.2.

☞ 137

Central to the discussion of gradient estimation is the interchange between differentiation and integration. The following theorem provides sufficient conditions.

Theorem 7.1.1 (Interchanging Differentiation and Integration) *Let the function $g(\mathbf{x};\boldsymbol{\theta})$ be differentiable at $\boldsymbol{\theta}_0 \in \mathbb{R}^k$. Denote the corresponding gradient by $\nabla_{\boldsymbol{\theta}}\, g(\mathbf{x};\boldsymbol{\theta}_0)$. We assume that as a function of $\mathbf{x}$ this gradient is integrable. If there exists a neighborhood Θ of $\boldsymbol{\theta}_0$ and an integrable function $M(\mathbf{x};\boldsymbol{\theta}_0)$ such that for all $\boldsymbol{\theta} \in \Theta$*

$$\frac{|g(\mathbf{x};\boldsymbol{\theta}) - g(\mathbf{x};\boldsymbol{\theta}_0)|}{\|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|} \leqslant M(\mathbf{x};\boldsymbol{\theta}_0)\ , \tag{7.2}$$

then

$$\nabla_{\boldsymbol{\theta}} \int g(\mathbf{x};\boldsymbol{\theta})\,\mathrm{d}\mathbf{x}\,\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} = \int \nabla_{\boldsymbol{\theta}}\, g(\mathbf{x};\boldsymbol{\theta}_0)\,\mathrm{d}\mathbf{x}\ . \tag{7.3}$$

Proof: Let

$$\psi(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\theta}_0) = \frac{g(\mathbf{x};\boldsymbol{\theta}) - g(\mathbf{x};\boldsymbol{\theta}_0) - (\boldsymbol{\theta}-\boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} g(\mathbf{x};\boldsymbol{\theta}_0)}{\|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|}\ .$$

Condition (7.2) implies that $|\psi(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\theta}_0)| \leqslant M(\mathbf{x};\boldsymbol{\theta}_0) + \|\nabla_{\boldsymbol{\theta}} g(\mathbf{x};\boldsymbol{\theta}_0)\|$ for all $\boldsymbol{\theta} \in \Theta$. Moreover, by the existence of the gradient at $\boldsymbol{\theta}_0$, we have that $\psi(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\theta}_0) \to 0$ as $\boldsymbol{\theta} \to \boldsymbol{\theta}_0$. Therefore, by the dominated convergence theorem, $\int \psi(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\theta}_0)\,\mathrm{d}\mathbf{x} \to 0$ as $\boldsymbol{\theta} \to \boldsymbol{\theta}_0$, which shows that (7.3) must hold.

An important special case where differentiation and integration can be interchanged arises in the theory of (natural) exponential families as summarized by the following theorem.

Theorem 7.1.2 (Interchange in Exponential Families) *For any function ϕ for which*

$$\int \phi(\mathbf{x})\,\mathrm{e}^{\boldsymbol{\eta}^\top \mathbf{t}(\mathbf{x})}\,\mathrm{d}\mathbf{x} < \infty\ ,$$

the integral as a function of $\boldsymbol{\eta}$ has partial derivatives of all orders, for all $\boldsymbol{\eta}$ in the interior of the natural parameter space. Moreover, these derivatives can be obtained by differentiating under the integral sign. That is,

$$\nabla_{\boldsymbol{\eta}} \int \phi(\mathbf{x})\,\mathrm{e}^{\boldsymbol{\eta}^\top \mathbf{t}(\mathbf{x})}\,\mathrm{d}\mathbf{x} = \int \phi(\mathbf{x})\,\mathbf{t}(\mathbf{x})\,\mathrm{e}^{\boldsymbol{\eta}^\top \mathbf{t}(\mathbf{x})}\,\mathrm{d}\mathbf{x}\ .$$

Remark 7.1.1 (Distributional and Structural Parameters) In many cases it is possible to switch from structural to distributional parameters and vice versa by making appropriate transformations. We discuss two common situations for the case where $\mathbf{x} = x$ is scalar (generalizations to the multidimensional case are straightforward).

1. *Push-out method:* Suppose the estimation problem involves only structural parameters; for example,

$$\ell(\boldsymbol{\theta}) = \int H(x;\boldsymbol{\theta})\, f(x)\, \mathrm{d}x \,. \tag{7.4}$$

The **push-out method** involves a (problem-dependent) change of variable $y = a(x;\boldsymbol{\theta})$ such that the transformed integral has the form

$$\ell(\boldsymbol{\theta}) = \int L(y)\, g(y;\boldsymbol{\theta})\, \mathrm{d}y \,. \tag{7.5}$$

In other words, the parameter $\boldsymbol{\theta}$ is "pushed-out" into the pdf g.

As a simple example consider the estimation of $\ell(\theta) = \mathbb{E}\exp(-X^\theta),\ \theta > 0$, where $X \sim f(x)$ is a positive random variable. This is of the form (7.4) with $H(x;\theta) = \exp(-x^\theta)$. Defining $y = x^\theta$, $L(y) = \exp(-y)$, and

$$g(y;\theta) = f\big(y^{\frac{1}{\theta}}\big)\frac{1}{\theta}\, y^{\frac{1}{\theta}-1} = f(x)\,\frac{1}{\theta}\, x^{1-\theta} \,,$$

the structural problem (7.4) is transformed into a distributional one (7.5).

2. *Inverse-transform method:* The inverse-transform method for random variable generation can be used to convert distributional estimation problems into structural ones. In particular, suppose $\ell(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}} L(Y)$ is of the form (7.5), and $G(y;\boldsymbol{\theta})$ is the cdf of $Y \sim g(y;\boldsymbol{\theta})$. By the inverse-transform method we can write $Y = G^{-1}(X;\boldsymbol{\theta})$, where $X \sim \mathsf{U}(0,1)$. If we now define $H(X;\boldsymbol{\theta}) = L(G^{-1}(X;\boldsymbol{\theta}))$, then $\ell(\boldsymbol{\theta}) = \mathbb{E}H(X;\boldsymbol{\theta})$ is of the form (7.4) with $f(x) = \mathrm{I}_{\{0<x<1\}}$. ☞ 26

7.2 Finite Difference Method

Let the performance measure $\ell(\boldsymbol{\theta})$ depend on a parameter $\boldsymbol{\theta} \in \mathbb{R}^d$. Suppose $\widehat{\ell}(\boldsymbol{\theta})$ is an estimator of $\ell(\boldsymbol{\theta})$ obtained via simulation. A straightforward estimator of the i-th component of $\nabla\ell(\boldsymbol{\theta})$, that is, $\partial\ell(\boldsymbol{\theta})/\partial\theta_i$, is the **forward difference estimator**

$$\frac{\widehat{\ell}(\boldsymbol{\theta} + \mathbf{e}_i\,\delta) - \widehat{\ell}(\boldsymbol{\theta})}{\delta} \,,$$

where $\mathbf{e}_i$ denotes the i-th unit vector in $\mathbb{R}^d$ and $\delta > 0$. An alternative is the **central difference estimator**

$$\frac{\widehat{\ell}(\boldsymbol{\theta} + \mathbf{e}_i\,\delta/2) - \widehat{\ell}(\boldsymbol{\theta} - \mathbf{e}_i\,\delta/2)}{\delta} \,.$$

In general, both estimators are biased. The bias of the forward difference estimator is of the order $\mathcal{O}(\delta)$, whereas the bias of the central difference estimator is of the order $\mathcal{O}(\delta^2)$, so that the latter estimator is generally preferred. However, the forward difference estimator requires the evaluation of only $d+1$ points per estimate, while the central difference estimator requires evaluation of $2d$ points per estimate.

A good choice of δ depends on various factors. It should be small enough to reduce the bias, but large enough to keep the variance of the estimator small. The choice of δ is usually determined via a trial run in which the variance of the estimator is assessed.

It is important to implement the finite difference method using **common random variables** (CRVs). The idea is similar to that of antithetic random
☞ 105 variables and is as follows. As both terms in the difference estimator are produced via a simulation algorithm, they can be viewed as functions of a stream of independent uniform random variables. The important point to notice is that both terms *need not be independent*. In fact (considering only the central difference estimator), if we denote $Z_1 = \widehat{\ell}(\boldsymbol{\theta} - \mathbf{e}_i\,\delta/2)$ and $Z_2 = \widehat{\ell}(\boldsymbol{\theta} + \mathbf{e}_i\,\delta/2)$, then

$$\mathrm{Var}(Z_2 - Z_1) = \mathrm{Var}(Z_1) + \mathrm{Var}(Z_2) - 2\,\mathrm{Cov}(Z_1, Z_2)\;, \tag{7.6}$$

so that the variance of the estimator $(Z_2 - Z_1)/\delta$ can be reduced (relative to the independent case) by an amount $2\,\mathrm{Cov}(Z_1, Z_2)/\delta^2$, provided that Z_1 and Z_2 are positively correlated. This can be achieved in practice by taking the *same* random numbers in the simulation procedure to generate Z_1 and Z_2. Because δ is typically small, the correlation between Z_1 and Z_2 is typically close to 1, so that a large variance reduction can be achieved relative to the case where Z_1 and Z_2 are independent.

For the case where $\ell(\boldsymbol{\theta}) = \mathbb{E}Y = \mathbb{E}H(\mathbf{X};\boldsymbol{\theta}) = \mathbb{E}h(\mathbf{U};\boldsymbol{\theta})$, with $\mathbf{U} \sim \mathsf{U}(0,1)^d$, this leads to the following algorithm.

Algorithm 7.1 (Central Difference Estimation With CRVs)

1. *Generate* $\mathbf{U}_1, \ldots, \mathbf{U}_N \overset{\text{iid}}{\sim} \mathsf{U}(0,1)^d$.
2. *Let* $L_k = h(\mathbf{U}_k; \boldsymbol{\theta} - \mathbf{e}_i\,\delta/2)$ *and* $R_k = h(\mathbf{U}_k; \boldsymbol{\theta} + \mathbf{e}_i\,\delta/2)$, $k = 1, \ldots, N$.
3. *Compute the sample covariance matrix corresponding to the pairs* $\{(L_k, R_k)\}$*:*

$$C = \begin{pmatrix} \frac{1}{N-1}\sum_{k=1}^N (L_k - \bar{L})^2 & \frac{1}{N-1}\sum_{k=1}^N (L_k - \bar{L})(R_k - \bar{R}) \\ \frac{1}{N-1}\sum_{k=1}^N (L_k - \bar{L})(R_k - \bar{R}) & \frac{1}{N-1}\sum_{k=1}^N (R_k - \bar{R})^2 \end{pmatrix}.$$

4. *Estimate* $\partial\ell(\boldsymbol{\theta})/\partial\theta_i$ *via the central difference estimator*

$$\frac{\bar{R} - \bar{L}}{\delta}$$

with an estimated standard error of

$$SE = \frac{1}{\delta}\sqrt{\frac{C_{1,1} + C_{2,2} - 2\,C_{1,2}}{N}}\;.$$

Example 7.1 (Bridge Network via the Finite Difference Method) Consider the bridge network in Example 6.1 on Page 103. There, the performance measure ℓ is the expected length of the shortest path between the two end nodes and is of the form $\ell(\mathbf{a}) = h(\mathbf{U}; \mathbf{a})$, where $\mathbf{U} \sim \mathsf{U}(0,1)^5$ and $\mathbf{a}$ is a parameter vector. We wish to estimate the gradient of $\ell(\mathbf{a})$ at $\mathbf{a} = (1,2,3,1,2)^\top$. The central difference estimator is implemented in the MATLAB program below. A typical estimate for the gradient based on $N = 10^6$ samples is

$$\widehat{\nabla \ell}(\mathbf{a}) = \begin{pmatrix} 0.3977 \pm 0.0003 \\ 0.0316 \pm 0.0001 \\ 0.00257 \pm 0.00002 \\ 0.3981 \pm 0.0003 \\ 0.0316 \pm 0.0001 \end{pmatrix},$$

where the notation $x \pm \varepsilon$ indicates an estimate of x with an estimated standard error of ε. The above estimate suggests that the expected length of the shortest path is most sensitive to changes in the lengths of components 1 and 4, as is to be expected for these parameter values, because the shortest path is highly likely to consist of these two edges. Component 3 contributes very little to the shortest path and its gradient is close to 0.

```
%fd_bridge.m
N = 10^6;
a = [1,2,3,1,2];
delta = 10^-3;
u = rand(N,5);
for comp=1:5
  de = zeros(1,5);
  de(comp) = delta;
  L = h1(u,a - de/2);
  R = h1(u,a + de/2);
  c = cov(L,R);
  se = sqrt((c(1,1) + c(2,2) - 2*c(1,2))/N)/delta;
  gr = (mean(R) - mean(L))/delta;
  fprintf('%g pm %3.1e\n',  gr, se);
end
```

```
function out=h1(u,a)
N = size(u,1);
X = u.*repmat(a,N,1);
Path_1=X(:,1)+X(:,4);
Path_2=X(:,1)+X(:,3)+X(:,5);
Path_3=X(:,2)+X(:,3)+X(:,4);
Path_4=X(:,2)+X(:,5);
out=min([Path_1,Path_2,Path_3,Path_4],[],2);
```

7.3 Infinitesimal Perturbation Analysis

Infinitesimal perturbation analysis (IPA) concerns the estimation of the gradient of a performance measure $\ell(\boldsymbol{\theta})$ of the form (7.1) with only *structural* parameters. In particular, the objective is to estimate $\nabla \ell(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E} H(\mathbf{X}; \boldsymbol{\theta})$ for some function $H(\mathbf{x}; \boldsymbol{\theta})$ and $\mathbf{X} \sim f(\mathbf{x})$ through an interchange of the gradient and the expectation operator; that is,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E} H(\mathbf{X}; \boldsymbol{\theta}) = \mathbb{E} \nabla_{\boldsymbol{\theta}} H(\mathbf{X}; \boldsymbol{\theta}) \; . \tag{7.7}$$

Such an interchange is allowed under certain regularity conditions on H, see Theorem 7.1.1. If (7.7) holds, then the gradient can be estimated via crude Monte Carlo as

$$\widehat{\nabla \ell}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^{N} \nabla_{\boldsymbol{\theta}} H(\mathbf{X}_k; \boldsymbol{\theta}) \; , \tag{7.8}$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} f$. In contrast to the finite difference method, the IPA estimator is unbiased. Moreover, because the procedure is basically a crude Monte Carlo method, its rate of convergence is $\mathcal{O}(1/\sqrt{N})$. The IPA procedure is summarized in the following algorithm.

Algorithm 7.2 (IPA Estimation)

1. *Generate* $\mathbf{X}_1, \ldots, \mathbf{X}_N \overset{\text{iid}}{\sim} f$.
2. *Evaluate* $\nabla_{\boldsymbol{\theta}} H(\mathbf{X}_k; \boldsymbol{\theta})$, $k = 1, \ldots, N$ *and estimate the gradient of* $\ell(\boldsymbol{\theta})$ *via* (7.8). *Determine an approximate* $1 - \alpha$ *confidence interval as*

$$\left(\widehat{\nabla \ell}(\boldsymbol{\theta}) - z_{1-\alpha/2} \, S/\sqrt{N}, \;\; \widehat{\nabla \ell}(\boldsymbol{\theta}) + z_{1-\alpha/2} \, S/\sqrt{N} \right) ,$$

where S *is the sample standard deviation of* $\{\nabla_{\boldsymbol{\theta}} H(\mathbf{X}_k; \boldsymbol{\theta})\}$ *and* z_γ *denotes the* γ*-quantile of the* $\mathsf{N}(0,1)$ *distribution.*

Example 7.2 (Bridge Network via IPA) We consider the same derivative estimation problem as in Example 7.1, but deal with it via IPA. Denote the four possible paths in the bridge network by

$$\mathcal{P}_1 = \{1, 4\}, \qquad \mathcal{P}_2 = \{1, 3, 5\}, \qquad \mathcal{P}_3 = \{2, 3, 4\}, \qquad \mathcal{P}_4 = \{2, 5\} \; .$$

Then we can write

$$h(\mathbf{U}; \mathbf{a}) = \min_{k=1,\ldots,4} \sum_{i \in \mathcal{P}_k} a_i \, U_i \; . \tag{7.9}$$

Let $K \in \{1, 2, 3, 4\}$ be the (random) index of the minimum-length path; hence, $h(\mathbf{U}; \mathbf{a}) = \sum_{i \in \mathcal{P}_K} a_i \, U_i$. The partial derivatives of $h(\mathbf{U}; \mathbf{a})$ now follow immediately from (7.9):

$$\frac{\partial h(\mathbf{U}; \mathbf{a})}{\partial a_i} = \begin{cases} U_i & \text{if } K \in \mathcal{A}_i, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{A}_i$ is the set of indices of all paths that contain component i; that is,

$$\mathcal{A}_1 = \{1,2\}, \quad \mathcal{A}_2 = \{3,4\}, \quad \mathcal{A}_3 = \{2,3\}, \quad \mathcal{A}_4 = \{1,3\}, \quad \mathcal{A}_5 = \{2,4\} .$$

The IPA procedure is implemented in the MATLAB program below. A typical estimate for the gradient at $\mathbf{a} = (1,2,3,1,2)^\top$ is

$$\widehat{\nabla \ell}(\mathbf{a}) = \begin{pmatrix} 0.3980 \pm 0.0003 \\ 0.0316 \pm 0.0001 \\ 0.00255 \pm 0.00002 \\ 0.3979 \pm 0.0003 \\ 0.0316 \pm 0.0001 \end{pmatrix} ,$$

where the same notation is used as in Example 7.1. We see that the accuracy is similar to that of the central difference method with common random numbers. However, the IPA estimate is unbiased.

```
%ipabridge.m
N = 10^6;
a = [1,2,3,1,2];
A = [1,2;3,4;2,3;1,3;2,4];
u = rand(N,5);
for comp=1:5
    dh = zeros(N,1);
    [y,K] = HK(u,a);
    ind = find(K == A(comp,1) | K==A(comp,2));
    dh(ind) = u(ind,comp);
    gr = mean(dh);
    se = std(dh)/sqrt(N);
    fprintf('%g pm %3.1e\n',  gr, se);
end
```

```
function [y,K]=HK(u,a)
N = size(u,1);
X = u.*repmat(a,N,1);
Path_1=X(:,1)+X(:,4);
Path_2=X(:,1)+X(:,3)+X(:,5);
Path_3=X(:,2)+X(:,3)+X(:,4);
Path_4=X(:,2)+X(:,5);
[y,K] =min([Path_1,Path_2,Path_3,Path_4],[],2);
```

7.4 Score Function Method

In the **score function method**, also called the **likelihood ratio method**, the performance function $\ell(\boldsymbol{\theta})$ is assumed to be of the form (7.1) with only

distributional parameters. In particular, the objective is to estimate (in the continuous case) the gradient of

$$\ell(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}} H(\mathbf{X}) = \int H(\mathbf{x})\, f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x}$$

for some function H and pdf f (for the discrete case replace the integral with a sum). As with the IPA method the key is to interchange the gradient and the integral; that is,

$$\nabla_{\boldsymbol{\theta}} \int H(\mathbf{X})\, f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x} = \int H(\mathbf{X})\, \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x}\,, \tag{7.10}$$

which is allowed under quite general conditions (see Theorem 7.1.1). Note that the right-hand side of (7.10) can be written as

$$\begin{aligned}\int H(\mathbf{X})\, \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x} &= \int H(\mathbf{X})\, \frac{\nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})}{f(\mathbf{x};\boldsymbol{\theta})} f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x} \\ &= \int H(\mathbf{X})\, [\nabla_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta})]\, f(\mathbf{x};\boldsymbol{\theta})\, \mathrm{d}\mathbf{x} \\ &= \mathbb{E}_{\boldsymbol{\theta}} H(\mathbf{X})\, \mathcal{S}(\boldsymbol{\theta};\mathbf{X})\,,\end{aligned}$$

where $\mathcal{S}(\boldsymbol{\theta};\mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta})$ is the *score function* of f. Hence, if (7.10) holds, the gradient can be estimated via crude Monte Carlo as

$$\widehat{\nabla \ell}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \mathcal{S}(\boldsymbol{\theta};\mathbf{X}_k)\,, \tag{7.11}$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} f$. The score function estimator is unbiased, and, being the sample mean of iid random variables, achieves $\mathcal{O}(1/\sqrt{N})$ convergence.

Algorithm 7.3 (Gradient Estimation via the Score Function Method)

1. *Generate* $\mathbf{X}_1, \ldots, \mathbf{X}_N \overset{\text{iid}}{\sim} f(\cdot;\boldsymbol{\theta})$.
2. *Evaluate the scores* $\mathcal{S}(\boldsymbol{\theta};\mathbf{X}_k)$, $k = 1, \ldots, N$ *and estimate the gradient of* $\ell(\boldsymbol{\theta})$ *via* (7.11). *Determine an approximate* $1-\alpha$ *confidence interval as*

$$\left(\widehat{\nabla \ell}(\boldsymbol{\theta}) - z_{1-\alpha/2}\, \widehat{\sigma}/\sqrt{N},\ \widehat{\nabla \ell}(\boldsymbol{\theta}) + z_{1-\alpha/2}\, \widehat{\sigma}/\sqrt{N}\right),$$

where $\widehat{\sigma}$ *is the sample standard deviation of* $\{H(\mathbf{X}_k)\mathcal{S}(\boldsymbol{\theta};\mathbf{X}_k)\}$ *and* z_γ *denotes the* γ*-quantile of the* $\mathsf{N}(0,1)$ *distribution.*

Remark 7.4.1 (Higher-Order Derivatives) Higher-order derivatives of ℓ can be estimated in a similar fashion. Specifically, the r-th order derivative is given by

$$\nabla^r \ell(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}} \left[H(\mathbf{X})\, \mathcal{S}^{(r)}(\boldsymbol{\theta};\mathbf{X}) \right], \tag{7.12}$$

where

$$\mathcal{S}^{(r)}(\boldsymbol{\theta};\mathbf{x}) = \frac{\nabla^r_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})}{f(\mathbf{x};\boldsymbol{\theta})} \tag{7.13}$$

is the ***r*-th order score function**, $r = 0, 1, 2, \ldots$. In particular, $\mathcal{S}^{(0)}(\boldsymbol{\theta};\mathbf{x}) = 1$ (by definition), $\mathcal{S}^{(1)}(\boldsymbol{\theta};\mathbf{x}) = \mathcal{S}(\boldsymbol{\theta};\mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta})$, and $\mathcal{S}^{(2)}(\boldsymbol{\theta};\mathbf{x})$ can be represented as

$$\begin{aligned} \mathcal{S}^{(2)}(\boldsymbol{\theta};\mathbf{x}) &= \nabla_{\boldsymbol{\theta}} \mathcal{S}(\boldsymbol{\theta};\mathbf{x}) + \mathcal{S}(\boldsymbol{\theta};\mathbf{x})\, \mathcal{S}(\boldsymbol{\theta};\mathbf{x})^\top \\ &= \nabla^2_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta}) \left[\nabla_{\boldsymbol{\theta}} \ln f(\mathbf{x};\boldsymbol{\theta})\right]^\top . \end{aligned} \tag{7.14}$$

The higher-order $\nabla^r \ell(\boldsymbol{\theta})$, $r = 0, 1, \ldots$, can be estimated via simulation as

$$\widehat{\nabla^r \ell}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \mathcal{S}^{(r)}(\boldsymbol{\theta};\mathbf{X}_k) . \tag{7.15}$$

It follows that the function $\ell(\boldsymbol{\theta})$, and *all* the sensitivities $\nabla^r \ell(\boldsymbol{\theta})$ can be estimated from a single simulation, because in (7.12) all of them are expressed as expectations with respect to the same pdf, $f(\mathbf{x};\boldsymbol{\theta})$.

Example 7.3 (Bridge Network via the Score Function Method) Consider again the derivative estimation problem in Examples 7.1 and 7.2. As in Example 6.1 on Page 103 we can write

$$\ell(\mathbf{a}) = \int H(\mathbf{x}) f(\mathbf{x};\mathbf{a})\, \mathrm{d}\mathbf{x} ,$$

with $H(\mathbf{x}) = \min\{x_1 + x_4,\ x_1 + x_3 + x_5,\ x_2 + x_3 + x_4,\ x_2 + x_5\}$ and

$$f(\mathbf{x};\mathbf{a}) = \prod_{i=1}^{5} \frac{\mathrm{I}_{\{0<x_i<a_i\}}}{a_i} . \tag{7.16}$$

This is a typical example where the interchange of gradient and integral is *not* appropriate, because of the discontinuities at $a_1, \ldots, a_5$. However, the situation can easily be fixed by including a continuity correction. Taking the derivative with respect to a_1 gives,

$$\begin{aligned} \frac{\partial}{\partial a_1} \ell(\mathbf{a}) &= \frac{\partial}{\partial a_1} \int_0^{a_1} \left(\int_0^{a_2} \cdots \int_0^{a_5} H(\mathbf{x}) \frac{1}{a_1 \cdots a_5}\, \mathrm{d}x_2 \cdots \mathrm{d}x_5 \right) \mathrm{d}x_1 \\ &= \int_0^{a_2} \cdots \int_0^{a_5} H(a_1, x_2, \ldots, x_5) \frac{1}{a_1 \cdots a_5}\, \mathrm{d}x_2 \cdots \mathrm{d}x_5 \\ &\quad - \frac{1}{a_1} \int H(\mathbf{x}) f(\mathbf{x};\mathbf{a})\, \mathrm{d}\mathbf{x} \\ &= \frac{1}{a_1} \left(\mathbb{E} H(\mathbf{X}^*) - \mathbb{E} H(\mathbf{X}) \right) , \end{aligned} \tag{7.17}$$

where $\mathbf{X} \sim f(\mathbf{x};\mathbf{a})$ and $\mathbf{X}^* \sim f(\mathbf{x};\mathbf{a} \,|\, x_1 = a_1)$. Both $\mathbb{E} H(\mathbf{X}^*)$ and $\mathbb{E} H(\mathbf{X})$ or $\mathbb{E}[H(\mathbf{X}^*) - H(\mathbf{X})]$ can easily be estimated via Monte Carlo. The other partial derivatives follow by symmetry.

The following MATLAB program implements the procedure. The results are similar to those of the IPA and finite difference methods.

```
%sfbridge.m
N = 10^6;
a = [1,2,3,1,2];
u = rand(N,5);
for comp=1:5
    X = u.*repmat(a,N,1);
    hx = H(X);
    X(:,comp) = a(comp);
    hxs = H(X);
    R = (-hx + hxs)/a(comp);
    gr = mean(R);
    se = std(R)/sqrt(N);
    fprintf('%g pm %3.1e\n',  gr, se);
end
```

```
function out=H(X)
Path_1=X(:,1)+X(:,4);
Path_2=X(:,1)+X(:,3)+X(:,5);
Path_3=X(:,2)+X(:,3)+X(:,4);
Path_4=X(:,2)+X(:,5);

out=min([Path_1,Path_2,Path_3,Path_4],[],2);
```

7.4.1 Score Function Method With Importance Sampling

By combining the score function method with importance sampling one can estimate the derivatives $\nabla^r \ell(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}[H(\mathbf{X})\, \mathcal{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X})]$ simultaneously for several values of $\boldsymbol{\theta} \in \Theta$, using a single simulation run. The idea is as follows. Let $g(\mathbf{x})$ be an importance sampling density. Then $\nabla^r \ell(\boldsymbol{\theta})$ can be written as

$$\nabla^r \ell(\boldsymbol{\theta}) = \mathbb{E}_g[H(\mathbf{X})\, \mathcal{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X})\, W(\mathbf{X}; \boldsymbol{\theta})] \,, \tag{7.18}$$

where

$$W(\mathbf{x}; \boldsymbol{\theta}) = \frac{f(\mathbf{x}; \boldsymbol{\theta})}{g(\mathbf{x})} \tag{7.19}$$

is the likelihood ratio of $f(\mathbf{x}; \boldsymbol{\theta})$ and $g(\mathbf{x})$. The importance sampling estimator of $\nabla^r \ell(\boldsymbol{\theta})$ can be written as

$$\widehat{\nabla^r \ell}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \mathcal{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X}_k)\, W(\mathbf{X}_k; \boldsymbol{\theta}) \,, \tag{7.20}$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} g$. Note that $\widehat{\nabla^r \ell}(\boldsymbol{\theta})$ is an unbiased estimator of $\nabla^r \ell(\boldsymbol{\theta})$ for *all* $\boldsymbol{\theta}$. This means that by varying $\boldsymbol{\theta}$ and keeping g fixed we can, in principle, estimate the whole *response surface* $\{\nabla^r \ell(\boldsymbol{\theta}), \boldsymbol{\theta} \in \Theta\}$ without bias from a single simulation.

Often the importance sampling distribution is chosen in the *same* class of distributions as the original one. That is, $g(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}_0)$, for some $\boldsymbol{\theta}_0 \in \Theta$. If we denote the importance sampling estimator of $\ell(\boldsymbol{\theta})$ for a given $\boldsymbol{\theta}_0$ by $\widehat{\ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0)$, that is,

$$\widehat{\ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, W(\mathbf{X}_k; \boldsymbol{\theta}; \boldsymbol{\theta}_0) \;, \tag{7.21}$$

with $W(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\theta}_0) = f(\mathbf{x}; \boldsymbol{\theta}) / f(\mathbf{x}; \boldsymbol{\theta}_0)$, and the estimators in (7.20) by $\widehat{\nabla^r \ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0)$, then

$$\widehat{\nabla^r \ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = \nabla^r_{\boldsymbol{\theta}}\, \widehat{\ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \mathbb{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X}_k)\, W(\mathbf{X}_k; \boldsymbol{\theta}; \boldsymbol{\theta}_0) \;. \tag{7.22}$$

Thus, the estimators of the sensitivities are simply the sensitivities of the estimators.

For a given importance sampling pdf $f(\mathbf{x}; \boldsymbol{\theta}_0)$, the algorithm for estimating the sensitivities $\nabla^r \ell(\boldsymbol{\theta})$, $r = 0, 1, \ldots$, for multiple values of $\boldsymbol{\theta}$ from a single simulation run, is as follows.

Algorithm 7.4 (Gradient Estimation via the Score Function Method)

1. Generate a sample $\mathbf{X}_1, \ldots, \mathbf{X}_N \overset{\text{iid}}{\sim} f(\cdot; \boldsymbol{\theta}_0)$.
2. Calculate the sample performance $H(\mathbf{X}_k)$ and the scores $\mathbb{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X}_k)$, $k = 1, \ldots, N$, for the desired parameter $\boldsymbol{\theta}$.
3. Calculate $\nabla^r_{\boldsymbol{\theta}}\, \widehat{\ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0)$ according to (7.22).

Confidence regions for $\nabla^r \ell(\boldsymbol{\theta})$ can be obtained by standard statistical techniques. In particular, $N^{1/2}\big[\nabla^r_{\boldsymbol{\theta}}\, \widehat{\ell}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) - \nabla^r \ell(\boldsymbol{\theta})\big]$ converges in distribution to a multivariate normal random vector with mean zero and covariance matrix

$$\text{Cov}_{\boldsymbol{\theta}_0}(H\, \mathbb{S}^{(r)}\, W) = \mathbb{E}_{\boldsymbol{\theta}_0}\left[H^2\, W^2\, \mathbb{S}^{(r)} \mathbb{S}^{(r)\,\top}\right] - [\nabla^r \ell(\boldsymbol{\theta})][\nabla^r \ell(\boldsymbol{\theta})]^\top, \tag{7.23}$$

using the abbreviations $H = H(\mathbf{X})$, $\mathbb{S}^{(r)} = \mathbb{S}^{(r)}(\boldsymbol{\theta}; \mathbf{X})$ and $W = W(\mathbf{X}; \boldsymbol{\theta}, \boldsymbol{\theta}_0)$.

Example 7.4 (Gradient Estimation for the Bridge Network) We return to Example 6.1. Let the lengths $X_2, \ldots, X_5$ of links $2, \ldots, 5$ be independent and uniformly distributed on $(0,2), (0,3), (0,1)$, and $(0,2)$, respectively. But let $X_1 \sim \mathsf{Exp}(\theta)$, independently of the other X_i. Hence, the only change in the setting of Example 6.1 is that the first component has an $\mathsf{Exp}(\theta)$ distribution, rather than a $\mathsf{U}(0,1)$ distribution. Denote the expected length of the shortest path by $\ell(\theta)$. Suppose the simulation is carried out under ☞ 103

$\theta = \theta_0 = 3$. We wish to estimate the derivative $\nabla \ell(\theta;\theta_0)$ for all θ in the neighborhood of θ_0. This is achieved by differentiating $\widehat{\ell}(\theta;\theta_0)$, giving

$$\begin{aligned}\widehat{\nabla \ell}(\theta;\theta_0) = \nabla \widehat{\ell}(\theta;\theta_0) &= \nabla \frac{\theta_0}{\theta\, N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \mathrm{e}^{X_{k1}(1/\theta_0 - 1/\theta)} \\ &= \frac{\theta_0}{\theta^3 N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, (X_{k1} - \theta)\, \mathrm{e}^{X_{k1}(1/\theta_0 - 1/\theta)} \\ &= \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, W(\mathbf{X}_k;\theta;\theta_0)\, \mathcal{S}(\theta; X_{k1}) \;,\end{aligned}$$

with $\mathcal{S}(\theta;x) = (x-\theta)/\theta^2$ being the score function corresponding to the $\mathsf{Exp}(1/\theta)$ distribution. The estimate of the derivative curve is given in Figure 7.1.

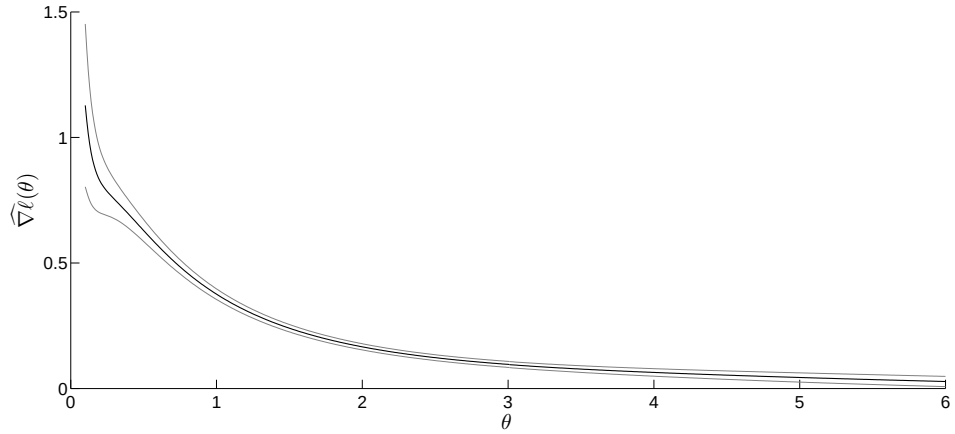


Figure 7.1: Estimates of the gradient of $\ell(\theta)$ with 95% confidence bounds.

The following MATLAB program implements the gradient estimation procedure.

```
%gradresponsesurfis.m
N = 10000;
theta0 = 3;
a = [theta0,2,3,1,2];
u = rand(N,5);
X = u.*repmat(a,N,1);
X(:,1) = -log(u(:,1))*theta0;
W = zeros(N,1);
Sc = zeros(N,1);
HX = H(X);
theta = 0.1:0.01:theta0*2;
num = numel(theta);
gradell = zeros(1,num);
```

```
gradellL = zeros(1,num);
gradellU = zeros(1,num);
stgradell = zeros(1,num);
for i=1:num
    th = theta(i);
    Sc = (-th + X(:,1))/th^2;
    W = (exp(-(X(:,1)/th))/th)./(exp(-(X(:,1)/theta0))/theta0);
    HWS = H(X).*W.*Sc;
    gradell(i) = mean(HWS);
    stgradell(i) = std(HWS);
    gradellL(i)= gradell(i) - stgradell(i)/sqrt(N)*1.95;
    gradellU(i)= gradell(i) + stgradell(i)/sqrt(N)*1.95;
end
plot(theta,gradell, theta, gradellL, theta, gradellU)
```

Chapter 8

Randomized Optimization

In this chapter we discuss optimization methods that have randomness as a core ingredient. Such randomized algorithms can be useful for solving optimization problems with many local optima and complicated constraints, possibly involving a mix of continuous and discrete variables. Randomized algorithms are also used to solve *noisy* optimization problems, in which the objective function is unknown and has to be obtained via Monte Carlo simulation.

We consider randomized optimization methods for both noisy and deterministic problems, including *stochastic approximation*, the *stochastic counterpart method*, *simulated annealing*, and *evolutionary algorithms*. Another such method, the *cross-entropy method*, is discussed in more detail in Chapter 9.

We refer to Chapter 7 for gradient estimation techniques. ☞ 123

Throughout this chapter we use the letter S to denote the objective function.

8.1 Stochastic Approximation

Suppose we have a minimization problem on $\mathscr{X} \subseteq \mathbb{R}^n$ of the form

$$\min_{\mathbf{x} \in \mathscr{X}} S(\mathbf{x}) \,, \tag{8.1}$$

where S is an unknown function of the form $\mathbb{E}\widetilde{S}(\mathbf{x}, \boldsymbol{\xi})$, with $\boldsymbol{\xi}$ a random vector and $\widetilde{S}$ a known function. A typical example is where $S(\mathbf{x})$ is the (usually unknown) expected performance measure from a Monte Carlo simulation. Such a problem is said to be a **noisy** optimization problem, as typically only realizations of $\widetilde{S}(\mathbf{x}, \boldsymbol{\xi})$ can be observed.

Because the gradient ∇S is unknown, one cannot directly apply classical optimization methods. The **stochastic approximation method** mimics simple gradient descent by replacing a deterministic gradient with a random approximation. More generally, one can approximate a subgradient instead of the gradient. It is assumed that an estimate of the gradient of S is available at any point $\mathbf{x} \in \mathscr{X}$. We denote such an estimate by $\widehat{\nabla S}(\mathbf{x})$. There are several established ways of obtaining $\widehat{\nabla S}(\mathbf{x})$. These include the finite difference method, infinitesimal perturbation analysis, the score function method, and the method of weak derivatives — see Chapter 7, where S is replaced by ℓ and $\mathbf{x}$ by $\boldsymbol{\theta}$. ☞ 123

In direct analogy to gradient descent methods, the stochastic approximation method produces a sequence of iterates, starting with some $\mathbf{x}_1 \in \mathscr{X}$, via

$$\mathbf{x}_{t+1} = \Pi_{\mathscr{X}} \left(\mathbf{x}_t - \beta_t \widehat{\nabla S}(\mathbf{x}_t) \right) , \tag{8.2}$$

where $\beta_1, \beta_2, \ldots$ is a sequence of strictly positive step sizes and $\Pi_{\mathscr{X}}$ is a projection operator that takes a point in $\mathbb{R}^n$ and returns a closest (typically in Euclidean distance) point in $\mathscr{X}$, ensuring that iterates remain feasible. That is, for any $\mathbf{y} \in \mathbb{R}^n$, $\Pi_{\mathscr{X}}(\mathbf{y}) \in \operatorname{argmin}_{\mathbf{z} \in \mathscr{X}} \|\mathbf{z} - \mathbf{y}\|$. Naturally, if $\mathscr{X} = \mathbb{R}^n$, then $\Pi_{\mathscr{X}}(\mathbf{y}) = \mathbf{y}$. A generic stochastic approximation algorithm is as follows.

Algorithm 8.1 (Stochastic Approximation)

1. *Initialize* $\mathbf{x}_1 \in \mathscr{X}$. *Set* $t = 1$.
2. *Obtain an estimated gradient* $\widehat{\nabla S}(\mathbf{x}_t)$ *of* S *at* $\mathbf{x}_t$.
3. *Determine a step size* β_t.
4. *Set* $\mathbf{x}_{t+1} = \Pi_{\mathscr{X}} \left(\mathbf{x}_t - \beta_t \widehat{\nabla S}(\mathbf{x}_t) \right)$.
5. *If a stopping criterion is met, stop; otherwise, set* $t = t + 1$ *and repeat from Step 2.*

For an arbitrary deterministic positive sequence $\beta_1, \beta_2, \ldots$ such that

$$\sum_{t=1}^{\infty} \beta_t = \infty, \quad \sum_{t=1}^{\infty} \beta_t^2 < \infty ,$$

the random sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots$ converges in the mean square sense to the minimizer $\mathbf{x}^*$ of $S(\mathbf{x})$ under certain regularity conditions.

We now present one of the simplest convergence theorems from Shapiro.

Theorem 8.1.1 (Convergence of Stochastic Approximation) *Suppose the following conditions are satisfied:*

1. *The feasible set* $\mathscr{X} \subset \mathbb{R}^n$ *is convex, nonempty, closed, and bounded.*
2. $\Pi_{\mathscr{X}}$ *is the Euclidean projection operator.*
3. *The objective function* S *is well defined, finite valued, continuous, differentiable, and strictly convex in* $\mathscr{X}$ *with parameter* $\beta > 0$. *That is, there exists a* $\beta > 0$ *such that*

$$(\mathbf{y} - \mathbf{x})^\top (\nabla S(\mathbf{y}) - \nabla S(\mathbf{x})) \geqslant \beta \|\mathbf{y} - \mathbf{x}\|^2 \quad \textit{for all } \mathbf{x}, \mathbf{y} \in \mathscr{X} .$$

4. *The error in the stochastic gradient vector* $\widehat{\nabla S}(\mathbf{x})$ *possesses a bounded second moment. That is, for some* $K > 0$,

$$\mathbb{E} \|\widehat{\nabla S}(\mathbf{x})\|^2 \leqslant K^2 < \infty \quad \textit{for all } \mathbf{x} \in \mathscr{X} .$$

Then, if $\beta_t = c/t$ for $c > 1/(2\beta)$,

$$\mathbb{E}\,\|\mathbf{x}_t - \mathbf{x}^*\|^2 \leqslant \frac{Q(c)}{t}\,, \quad t = 1, 2, \ldots\,,$$

where

$$Q(c) = \max\{c^2 K^2\,(2c\beta - 1)^{-1}\,,\, \|\mathbf{x}_1 - \mathbf{x}^*\|^2\}\,,$$

with minimal Q attained by choosing $c = 1/\beta$. In other words, the expected error in terms of Euclidean distance of the iterates is of order $\mathcal{O}(t^{-1/2})$.

Moreover, if $\mathbf{x}^*$ is an interior point of $\mathscr{X}$ and if there is some constant $L > 0$ such that

$$\|\nabla S(\mathbf{y}) - \nabla S(\mathbf{x})\| \leqslant L\|\mathbf{y} - \mathbf{x}\| \quad \textit{for all } \mathbf{x}, \mathbf{y} \in \mathscr{X}\,,$$

(that is, $\nabla S(\mathbf{x})$ is uniformly Lipschitz continuous in $\mathscr{X}$), then

$$\mathbb{E}\,|S(\mathbf{x}_t) - S(\mathbf{x}^*)| \leqslant \frac{L\,Q(c)}{2t}\,, \quad t = 1, 2, \ldots\,.$$

In other words, the expected error in terms of Euclidean distance of the objective function values is of order $\mathcal{O}(t^{-1})$.

An attractive feature of the stochastic approximation method is its simplicity and ease of implementation in those cases where the projection $\Pi_{\mathscr{X}}$ can be easily computed. For example with **box-constraints**, where $\mathscr{X} = [a_1, b_1] \times \cdots \times [a_n, b_n]$, any component x_k of $\mathbf{x}$ is projected to a_k if $x_k < a_k$ and to b_k if $x_k > b_k$, and otherwise remains unchanged.

A weak point of the method is the ambiguity in choosing the step size sequence $\beta_1, \beta_2, \ldots$. Small step sizes lead to slow convergence and large step sizes may result in "zigzagging" behavior of the iterates. A commonly used choice is $\beta_t = c/t$ for some constant c, as suggested by Theorem 8.1.1. The practical performance of the algorithm using this step size rule depends crucially on c. This naturally leads to the idea of adaptively tuning this constant to the problem at hand.

If $\beta_t/\beta_{t+1} = 1 + o(\beta_t)$, as is the case when $\beta_t = 1/t^\gamma$ with $\gamma \in (0, 1)$, then the averaged iterate sequence defined by $\bar{\mathbf{x}}_t = \frac{1}{t}\sum_{k=1}^{t} \mathbf{x}_k$ tends to give better results than $\{\mathbf{x}_t\}$ itself. This is known as **Polyak averaging** or **iterate averaging**. One advantage here is that the algorithm will take larger step sizes than the $1/t$ case, speeding up the location of a solution.

When $\widehat{\nabla S}(\mathbf{x}_t)$ is an *unbiased* estimator of $\nabla S(\mathbf{x}_t)$ in (8.2) the stochastic approximation Algorithm 8.1 is referred to as the **Robbins–Monro** algorithm. When finite differences are used to estimate $\widehat{\nabla S}(\mathbf{x}_t)$ the resulting algorithm is known as the **Kiefer–Wolfowitz** algorithm. As noted in Section 7.2, the gradient estimate usually has a bias that depends on the length of the interval corresponding to the central or forward difference estimators. ☞ 125

In high dimensions the **random directions** procedure can be used instead of the usual Kiefer–Wolfowitz algorithm, reducing the number of function evaluations per gradient estimate to two. This can be achieved as follows. Let $\mathbf{D}_1, \mathbf{D}_2, \ldots$ be a sequence of random direction vectors in $\mathbb{R}^n$, typically satisfying (though these are not strictly required the following conditions.

- The vectors are iid and symmetrically distributed with respect to each of the coordinate axes, with $\mathbb{E}\boldsymbol{D}_t\boldsymbol{D}_t^\top = I$ and $\|\boldsymbol{D}_t\|^2 = n$.

For example, one can take each $\boldsymbol{D}_t$ distributed uniformly on the sphere of radius $\sqrt{n}$, or each $\boldsymbol{D}_t$ distributed uniformly on $\{-1, 1\}^n$ (that is, each component takes the values ± 1 with probability $1/2$). However, the random directions method can exhibit poor behavior if the number of iterations is not sufficiently large.

Example 8.1 (Noisy Optimization by Stochastic Approximation) We illustrate the stochastic approximation procedure via a simple problem of the form (8.1), with

$$\widetilde{S}(\mathbf{x}, \boldsymbol{\xi}) = \|\boldsymbol{\xi} - \mathbf{x}\|^2, \qquad \boldsymbol{\xi} \sim \mathsf{N}(\boldsymbol{\mu}, I) \; .$$

The function $S(\mathbf{x}) = \mathbb{E}\widetilde{S}(\mathbf{x}, \boldsymbol{\xi})$ has its minimum at $\mathbf{x}^* = \boldsymbol{\mu}$, with $S(\mathbf{x}^*) = n$. For this example we have $\nabla S(\mathbf{x}) = 2(\mathbf{x} - \boldsymbol{\mu})$. An unbiased (Robbins–Monro) estimator is

$$\widehat{\nabla S}(\mathbf{x})_{\mathrm{RM}} = \frac{1}{N}\sum_{k=1}^{N} 2(\mathbf{x} - \boldsymbol{\xi}_k) \, ,$$

where $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N \sim_{\mathrm{iid}} \mathsf{N}(\boldsymbol{\mu}, I)$. A central difference (Kiefer–Wolfowitz) estimator, with difference interval δ, is

$$(\widehat{\nabla S}(\mathbf{x})_{\mathrm{KW}})_i = \frac{1}{N}\sum_{k=1}^{N} \frac{\widetilde{S}(\mathbf{x} + \mathbf{e}_i\,\delta/2, \boldsymbol{\xi}_k) - \widetilde{S}(\mathbf{x} - \mathbf{e}_i\,\delta/2, \boldsymbol{\zeta}_k)}{\delta} \, , \quad i = 1, \ldots, n \; ,$$

☞ 125

where $\boldsymbol{\xi}_1, \boldsymbol{\zeta}_1, \ldots, \boldsymbol{\xi}_N, \boldsymbol{\zeta}_N \sim_{\mathrm{iid}} \mathsf{N}(\boldsymbol{\mu}, I)$. As observed in Section 7.2, the variance of this estimator can be reduced significantly by using common random variables. A practical way to do this here is to take $\boldsymbol{\zeta}_k = \boldsymbol{\xi}_k$ for each $k = 1, \ldots, N$.

Figure 8.1 illustrates the typical performance of the Robbins–Monro and Kiefer–Wolfowitz algorithms for this problem. The Kiefer–Wolfowitz algorithm is run with and without using common random variables. For each method we use 10^4 iterations. The problem dimension is $n = 100$ and $\boldsymbol{\mu} = (n, n-1, \ldots, 2, 1)^\top$. Each gradient estimate is computed using $N = 10$ independent trials. The MATLAB implementation is given below.

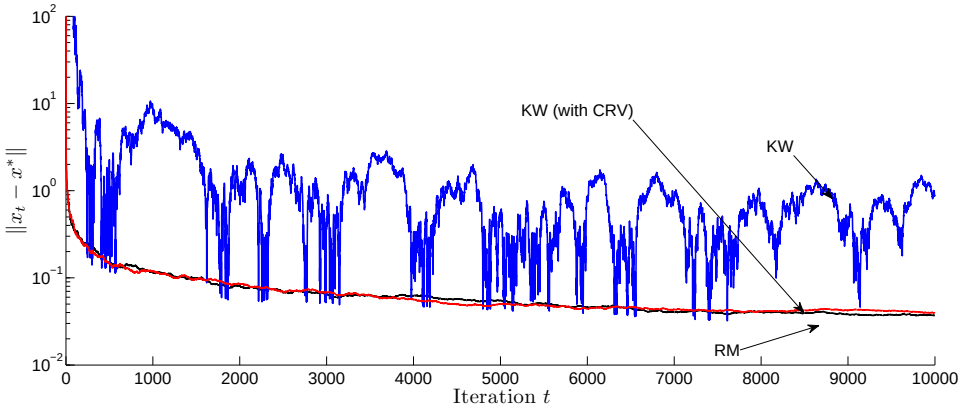


Figure 8.1: Typical convergence of Robbins–Monro and Kiefer–Wolfowitz algorithms (with and without making use of common random variables).

```
%StochApprox.m
maxits=10^4; % number of iterations
n=10^2; % dimension
N=10^1; % number of trials
mu=(n:-1:1); rmu=repmat(mu,N,1); % problem data
L=zeros(N,n); R=L; % allocate space for the c-diff. estimator

c=1; % constant for the step size
delta = 1; % constant for the FD sequence
betat=@(t) c./t; % step size functions
deltat=@(t) delta/t.^(1/6); % difference interval function

xrm=10.*n.*ones(1,n); % initial Robbins-Monro iterate
xkw=10.*n.*ones(1,n); % initial Kiefer-Wolfowitz iterate
xkwCRV=10.*n.*ones(1,n); % inital Kiefer-Wolfowitz iterate (CRV)

% allocate space for the convergence history of each iterate
rmhist=zeros(1,maxits);
kwhist=zeros(1,maxits);
kwCRVhist=zeros(1,maxits);
% compute initial distance to optimal solution
rmhist(1)=sqrt(sum((xrm-mu).^2));
kwhist(1)=sqrt(sum((xkw-mu).^2));
kwCRVhist(1)=sqrt(sum((xkwCRV-mu).^2));

t=1; % iteration Counter
while (t<maxits)
    % RM gradient est.
    xi=rmu+randn(N,n);
    grm=mean(2.*(repmat(xrm,N,1)-xi),1); % unbiased est.
```

```
    % KW gradient est.
    xiL=rmu+randn(N,n);
    xiR=rmu+randn(N,n);
    xkwN=repmat(xkw,N,1);
    e1=zeros(1,n);e1(1)=deltat(t)/2;
    ekN=repmat(e1,N,1);
    for k=1:n
        L(:,k)=sum((xiL-(xkwN+ekN)).^2,2);
        R(:,k)=sum((xiR-(xkwN-ekN)).^2,2);
        ekN=circshift(ekN,[0 1]);
    end
    gkw=mean((L-R)./deltat(t),1);
    % KW gradient est. with CRV
    xiL=rmu+randn(N,n);
    xiR=xiL; % practical CRV
    xkwCRVN=repmat(xkwCRV,N,1);
    for k=1:n
        L(:,k)=sum((xiL-(xkwCRVN+ekN)).^2,2);
        R(:,k)=sum((xiR-(xkwCRVN-ekN)).^2,2);
        ekN=circshift(ekN,[0 1]);
    end
    gkwCRV=mean((L-R)./deltat(t),1);
    % Update Iterates
    xrm=xrm-betat(t).*grm;
    xkw=xkw-betat(t).*gkw;
    xkwCRV=xkwCRV-betat(t).*gkwCRV;
    % increase counter and record new distance to optimum
    t=t+1;
    rmhist(t)=sqrt(sum((xrm-mu).^2));
    kwhist(t)=sqrt(sum((xkw-mu).^2));
    kwCRVhist(t)=sqrt(sum((xkwCRV-mu).^2));
end
% plot the results
tt=(1:1:(maxits));
figure,semilogy(tt,rmhist,'k-',tt,kwhist,'b-',tt,...
    kwCRVhist,'r-','Linewidth',1.5)
```

8.2 Stochastic Counterpart Method

Consider again the noisy optimization setting (8.1). The idea of the **stochastic counterpart method** (also called **sample average approximation**) is to replace the noisy optimization problem (8.1) with

$$\min_{\mathbf{x}\in\mathbb{R}^n} \widehat{S}(\mathbf{x}) \,, \tag{8.3}$$

where

$$\widehat{S}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \widetilde{S}(\mathbf{x}, \boldsymbol{\xi}_i)$$

is a sample average estimator of $S(\mathbf{x}) = \mathbb{E}\widetilde{S}(\mathbf{x}, \boldsymbol{\xi})$ on the basis of N iid samples $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N$.

A solution $\widehat{\mathbf{x}}^*$ to this sample average version is taken to be an estimator of a solution $\mathbf{x}^*$ to the original problem (8.1). Note that (8.3) is a *deterministic* optimization problem.

Example 8.2 (Stochastic Counterpart) Consider the following parametric optimization problem that arises when applying the CE method: given a family of densities $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathscr{V}\}$ and a target density g, locate the CE optimal parameter $\mathbf{v}^*$ that maximizes ☞ 155

$$\mathcal{D}(\mathbf{v}) \stackrel{\text{def}}{=} \mathbb{E}_g \ln f(\mathbf{Z};\mathbf{v}) = \mathbb{E}_p \left[\frac{g(\mathbf{Z})}{p(\mathbf{Z})} \ln f(\mathbf{Z};\mathbf{v})\right] ,$$

where p is any pdf that *dominates* g; that is, $p(\mathbf{z}) = 0 \Rightarrow g(\mathbf{z}) = 0$. Typically this optimization problem is difficult to solve, but one can consider solving the stochastic counterpart instead, here given by

$$\max_{\mathbf{v}\in\mathscr{V}} \widehat{\mathcal{D}}(\mathbf{v}) \stackrel{\text{def}}{=} \max_{\mathbf{v}\in\mathscr{V}} \frac{1}{N}\sum_{k=1}^{N} \frac{g(\mathbf{Z}_k)}{p(\mathbf{Z}_k)} \ln f(\mathbf{Z}_k;\mathbf{v}) , \tag{8.4}$$

where $\mathbf{Z}_1, \ldots, \mathbf{Z}_N \sim_{\text{iid}} p$. For various parametric families $\{f(\cdot;\mathbf{v})\}$ this proxy problem is solvable analytically, providing the key updating formulæ for the CE method.

As a particular instance, suppose f is a Cauchy density, given by

$$f(z;\mathbf{v}) = \frac{1}{\pi\sigma}\frac{1}{1+\left(\frac{z-\mu}{\sigma}\right)^2} , \quad \mathbf{v} = (\mu, \sigma) ,$$

and the target density is

$$g(z) \propto \left|\exp(-z^2)\cos(3\pi z)\right| \exp\left(-\frac{1}{2}\left(\frac{z-1}{\sqrt{2}}\right)^2\right) .$$

The standard Cauchy density $p(z) = 1/(\pi(1+z^2))$ dominates $g(z)$.

Figure 8.2 depicts the pdfs obtained by solving 100 independent instances of the stochastic counterpart (8.4) for this problem, using approximating sample sizes of $N = 10^3$ and $N = 10^5$, respectively.

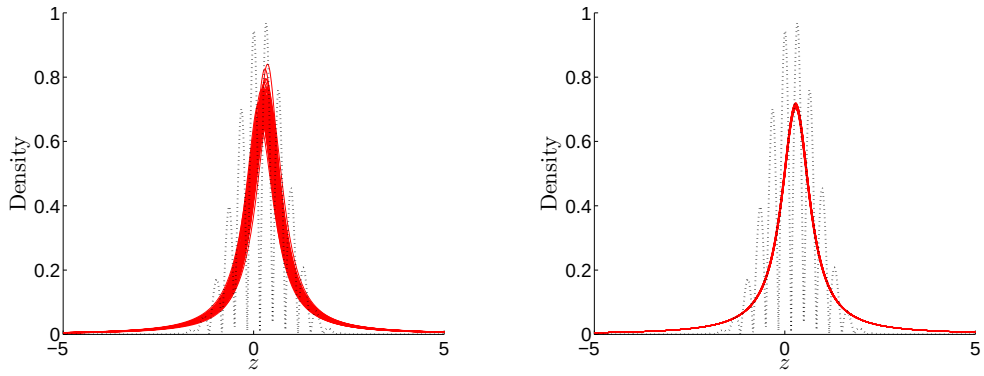


Figure 8.2: The pdfs of 100 independent solutions of the stochastic counterpart procedure using $N = 10^3$ (left) and $N = 10^5$ (right) samples. The dotted line is the target density g.

Figure 8.3 plots a typical sequence of estimates for μ and σ as a function of the sample size $N = 1, \ldots, 10^5$. The estimates of μ and σ obtained in this way strongly suggest convergence to an optimal value.

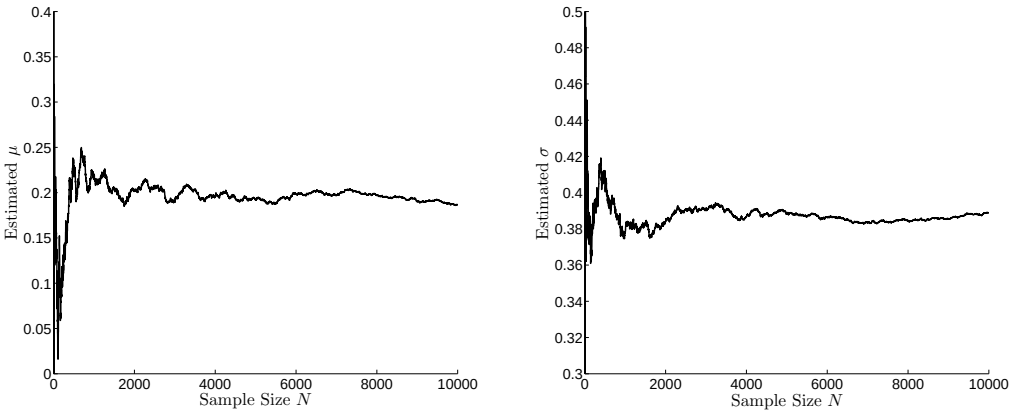


Figure 8.3: The estimates for μ and σ obtained by solving a sequence of stochastic counterpart problems for increasing N.

MATLAB code for the left panel in Figure 8.2 follows. Simply change the variable `N` to obtain the right panel. Code for Figure 8.3 is quite similar, and can be found on the Handbook website as `SCMb.m`.

```
%SCM.m
clear all
N=10^3; % Sample size
M=10^2; % Number of trials

g=@(Z) abs(exp(-Z.^2).*cos(3.*pi.*Z)).*...
    exp(-0.5.*((Z-1)./sqrt(2)).^2)./sqrt(2*pi*2);
```

```
h=@(Z,mu,sigma) (sigma>0)./(pi.*sigma.*(1+((Z-mu)./sigma).^2));
p=@(Z) 1./(pi.*(1+Z.^2)); % standard cauchy
f=@(x,Z) sum((g(Z)./p(Z)).*log(h(Z,x(1),x(2))));% S-Counterpart

approxnormg=.2330967533;% Approx. norm. const. for g
zz=linspace(-5,5,10^3);% Range to plot densities over
figure,hold on
for k=1:M
    Z=randn(N,1)./randn(N,1);% Z_1,...,Z_N are iid with pdf p
    sol=fminsearch(@(x) -f(x,Z),[1,2]);% Solve the SC
    plot(zz,h(zz,sol(1),sol(2)),'r-')
end
plot(zz,g(zz)./approxnormg,'k:','LineWidth',2) % Plot g
hold off
```

8.3 Simulated Annealing

Simulated annealing is a Markov chain Monte Carlo technique for approximately locating a global maximum of a given density $f(\mathbf{x})$. The idea is to create a sequence of points $\mathbf{X}_1, \mathbf{X}_2, \ldots$ that are approximately distributed according to pdfs $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots$ with $f_t(\mathbf{x}) \propto f(\mathbf{x})^{1/T_t}$, where $T_1, T_2, \ldots$ is a sequence of **temperatures** (known as the **annealing schedule**) that decreases to 0. If each $\mathbf{X}_t$ were sampled *exactly* from $f(\mathbf{x})^{1/T_t}$, then X_t would converge to a global maximum of $f(\mathbf{x})$ as $T_t \to 0$. However, in practice sampling is *approximate* and convergence to a global maximum is not assured. ☞ 87

A high-level simulated annealing algorithm is as follows.

Algorithm 8.2 (Simulated Annealing)

1. *Choose a starting state* $\mathbf{X}_0$ *and an initial temperature* T_0. *Set* $t = 1$.
2. *Select a temperature* $T_t \leqslant T_{t-1}$ *according to the annealing schedule.*
3. *Approximately generate a new state* $\mathbf{X}_t$ *from* $f_t(\mathbf{x}) \propto (f(\mathbf{x}))^{1/T_t}$.
4. *Set* $t = t + 1$ *and repeat from Step 2 until stopping.*

The most common application for simulated annealing is in optimization. In particular, consider the minimization problem

$$\min_{\mathbf{x} \in \mathscr{X}} S(\mathbf{x})$$

for some deterministic real-valued function $S(\mathbf{x})$. Define the **Boltzmann pdf** as

$$f(\mathbf{x}) \propto \mathrm{e}^{-S(\mathbf{x})}, \quad \mathbf{x} \in \mathscr{X} \ .$$

For $T > 0$ close to 0 the global maximum of $f(\mathbf{x})^{1/T} \propto \exp(-S(\mathbf{x})/T)$ is close to the global minimum of $S(\mathbf{x})$. Hence, by applying simulated annealing to the Boltzmann pdf, one can also minimize $S(\mathbf{x})$. Maximization problems can be handled in a similar way, by using a Boltzmann pdf $f(\mathbf{x}) \propto \exp(S(\mathbf{x}))$. Note that this may not define a valid pdf if the exponential terms are not normalizable.

There are many different ways to implement simulated annealing algorithms, depending on (1) the choice of Markov chain Monte Carlo sampling algorithm, (2) the length of the Markov chain between temperature updates, and (3) the annealing schedule. A popular annealing schedule is **geometric cooling**, where $T_t = \beta T_{t-1}$, $t = 1, 2, \ldots$, for a given initial temperature T_0 and a **cooling factor** $\beta \in (0, 1)$. Appropriate values for T_0 and β and are problem-dependent, and this has traditionally required tuning on the part of the user.

The following algorithm describes a popular simulated annealing framework
☞ 89
for minimization, which uses a random walk sampler; that is, a Metropolis–Hastings sampler with a symmetric proposal distribution. Note that the temperature is updated after a *single* step of the Markov chain.

Algorithm 8.3 (Simulated Annealing for Minimization)

1. *Initialize the starting state* $\mathbf{X}_0$ *and temperature* T_0. *Set* $t = 1$.
2. *Select a temperature* $T_t \leqslant T_{t-1}$ *from the annealing schedule.*
3. *Generate a candidate state* $\mathbf{Y}$ *from the symmetric proposal density* $q(\mathbf{Y} \,|\, \mathbf{X}_t) = q(\mathbf{X}_t \,|\, \mathbf{Y})$.
4. *Compute the acceptance probability*

$$\alpha(\mathbf{X}_t, \mathbf{Y}) = \min\left\{ \mathrm{e}^{-\frac{(S(\mathbf{Y}) - S(\mathbf{X}_t))}{T_t}},\, 1 \right\} .$$

   Generate $U \sim \mathsf{U}(0, 1)$ *and set*

$$\mathbf{X}_{t+1} = \begin{cases} \mathbf{Y} & \text{if } U \leqslant \alpha(\mathbf{X}_t, \mathbf{Y}) , \\ \mathbf{X}_t & \text{if } U > \alpha(\mathbf{X}_t, \mathbf{Y}) . \end{cases}$$

5. *Set* $t = t + 1$ *and repeat from Step 2 until a stopping criterion is met.*

Example 8.3 (Continuous Optimization by Simulated Annealing)
We illustrate the simulated annealing Algorithm 8.3 by applying it to the minimization of the *trigonometric function*. For n-dimensional $\mathbf{x}$, this function is given by

$$S(\mathbf{x}) = 1 + \sum_{i=1}^{n} \left(8 \sin^2(\eta (x_i - x_i^*)^2) + 6 \sin^2(2\eta (x_i - x_i^*)^2) + \mu (x_i - x_i^*)^2 \right) ,$$

and has minimal value of 1 at $\mathbf{x} = \mathbf{x}^*$. In this example, we choose $n = 10$, $\mathbf{x}^* = (10, \ldots, 10)$, $\eta = 0.8$, and $\mu = 0.1$.

In order to implement the method with Metropolis–Hastings sampling, there are four ingredients we must specify: (1) an appropriate initialization of the algorithm; (2) an annealing schedule $\{T_t\}$; (3) a proposal pdf q; and (4) a stopping criterion.

For initialization, we set $T_0 = 10$ and draw $\mathbf{X}_0$ uniformly from the n-dimensional hypercube $[-50, 50]^n$. We use a geometric cooling scheme with cooling factor $\beta = 0.99999$. The (symmetric) proposal distribution (starting from $\mathbf{x}$) is taken to be $\mathsf{N}(\mathbf{x}, \sigma^2 I)$, with $\sigma = 0.75$. The algorithm is set to stop after a 10^6 iterations. The MATLAB code is given below.

```
%SA.m
% Initialization
n=10; % dimension of the problem
beta=0.99999; % Factor in geometric cooling
sigma=ones(1,n).*0.75; % Variances for the proposal
N=1; %Number of steps to perform of MH
maxits=10^6; % Number of iterations
xstar=10.*ones(1,n); eta=0.8; mu=0.1;
S=@(X) 1+sum(mu.*(X-xstar).^2+6.*...
 (sin(2.*eta.*(X-xstar).^2)).^2+8.*(sin(eta.*(X-xstar).^2)).^2);
T=10; % Initial temperature
a=-50;b=50; X=(b-a).*rand(1,n)+a; %Initialize X
Sx=S(X); % Score the initial sample

t=1; % Initialize iteration counter
while (t<=maxits)
    T=beta*T; % Select New Temperature
    % Generate New State
    it=1;
    while (it<=N)
        Y=X+sigma.*randn(1,n);
        Sy=S(Y);
        alpha=min(exp(-(Sy-Sx)/T),1);
        if rand<=alpha
           X=Y; Sx=Sy;
        end
        it=it+1;
    end
    t=t+1; % Increment Iteration
end
[X,Sx,T] % Display final state, score, and temperature
```

For this example, we can sample exactly from the Boltzmann distribution via a straightforward application of *acceptance–rejection* (see Section 2.1.5). ☞ 38 As a consequence, we can see precisely how well our approximate sampling performs with respect to the ideal case.

In Figure 8.4, the average performance per iteration over 10 independent trials is plotted for both the approximate Metropolis–Hastings sampling from the Boltzmann pdf used above and exact sampling from the Boltzmann pdf via acceptance–rejection. For these parameters, the approximate scheme typically fails to locate the optimal solution.

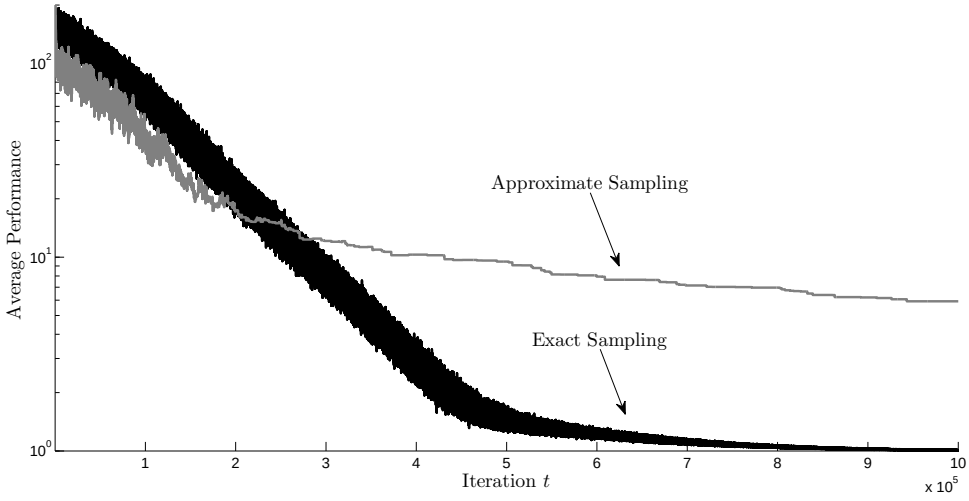


Figure 8.4: Average performance per iteration of exact and approximate sampling from the Boltzmann pdf, over 10 independent trials.

The MATLAB code used for this figure can be found on the Handbook website as `SAnnealing_Multi_SA.m`

8.4 Evolutionary Algorithms

Evolutionary algorithms refer to any metaheuristic framework that is inspired by the process of natural evolution. An algorithm of this type begins with a **population** $\mathscr{P}$ of **individuals** $\mathbf{x}$: objects such as points in $\mathbb{R}^n$, paths in a graph, etc. The population "evolves" from generation to generation in two stages. First, some **selection** mechanism is applied to create a new population. Second, some **alteration** mechanism is applied to the newly created population.

The objective is to create a population of individuals with superior qualities with respect to some performance measure(s) on the population. The simplest example is where $\mathscr{P}$ is a collection of n-dimensional points $\mathbf{x}$ and the goal is to minimize some objective function $S(\mathbf{x})$. In this case the **evaluation** of $\mathscr{P}$ corresponds to calculating $S(\mathbf{x})$ for all $\mathbf{x} \in \mathscr{P}$. Typically this information is used in the selection phase, for instance by only permitting the best performing 10% to be involved in creating the new generation.

The general framework for an evolutionary algorithm is summarized next.

Algorithm 8.4 (Generic Evolutionary Algorithm)

1. *Set $t = 0$. Initialize a population of individuals $\mathscr{P}_t$. Evaluate $\mathscr{P}_t$.*
2. *Select a new population $\mathscr{P}_{t+1}$ from $\mathscr{P}_t$.*
3. *Alter the population $\mathscr{P}_{t+1}$.*
4. *Evaluate $\mathscr{P}_{t+1}$.*
5. *If a stopping criterion is met, stop; otherwise, set $t = t + 1$ and repeat from Step 2.*

There are many well-known heuristic algorithms under the umbrella of evolutionary algorithms. We will discuss three in particular: genetic algorithms, differential evolution, and estimation of distribution algorithms.

8.4.1 Genetic Algorithms

The **genetic algorithm** metaheuristic is traditionally applied to discrete optimization problems. Individuals in the population are vectors, coded to represent potential solutions to the optimization problem. Each individual is ranked according to a fitness criterion (typically just the objective function value associated with that individual). A new population is then formed as children of the previous population. This is often the result of **cross-over** and **mutation** operations applied to the fittest individuals.

Suppose that individuals in the population are n-dimensional binary vectors $\mathbf{x}$, and the goal is to minimize some objective function $S(\mathbf{x})$. A possible cross-over mechanism in this case is **one-point crossover**: given two parents $\mathbf{x}$ and $\mathbf{y}$, and a random location r between 0 and n, create a new individual $\mathbf{z} = (x_1, \ldots, x_r, y_{r+1}, \ldots, y_n)$ whose first r components are copied from the first parent and the remaining $n - r$ components from the second parent.

Determining the M "fittest" individuals could be via **tournament selection**. In basic tournament selection with tournaments of size K, this involves selecting K individuals uniformly from the population, and selecting the individual with the lowest objective function value as the winner. The winner then joins the reproduction pool. This process is repeated M times, until the desired number of fittest individuals is selected.

A typical binary encoded genetic algorithm is as follows.

Algorithm 8.5 (Binary Encoded Genetic Algorithm)

1. *Set $t = 0$. Initialize a population of individuals $\mathscr{P}_t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_N^t\}$ via uniform sampling over $\{0, 1\}^n$. Evaluate $\mathscr{P}_t$.*
2. *Construct a* **reproduction pool** *$\mathscr{R}_t$ of individuals from $\mathscr{P}_t$ via tournament selection.*
3. **Combine** *individuals in the reproduction pool to obtain an intermediate population $\mathscr{C}_t$ via one-point crossover.*

4. **Mutate** *the intermediate population* $\mathscr{C}_t$ *by flipping each component of each binary vector independently with probability* $p = 1/n$. *Denote the resulting population by* $\mathscr{S}_t$.

5. **Create** *the new generation as* $\mathscr{P}_{t+1} = \mathscr{S}_t$. *Evaluate* $\mathscr{P}_{t+1}$.

6. *If a stopping criterion is met, stop; otherwise, set* $t = t + 1$ *and repeat from Step 2.*

Example 8.4 (Genetic Algorithm for the Satisfiability Problem) We illustrate the binary encoded genetic Algorithm 8.5 by applying it to solving the *satisfiability problem* (SAT). The problem is to find a binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ which, given a set of m clause functions $c_j(\cdot)$, satisfies all of them. Each clause function returns $c_j(\mathbf{x}) = 1$ if $\mathbf{x}$ satisfies clause j and $c_j(\mathbf{x}) = 0$ otherwise.

In Algorithm 8.5, we select a population and reproduction pool size of $N = 10^4$, a tournament size of $K = 2$ (binary tournament selection), and a we run the algorithm for 10^3 iterations.

In Figure 8.5, the scores of the best and worst performing individuals are plotted for a typical algorithm run on the difficult problem `F34-5-23-31` from `http://www.is.titech.ac.jp/~watanabe/gensat/a2/index.html`. Note that there are 361 clauses and 81 literals, and that the algorithm in this case locates solutions that satisfy at most 359 clauses.

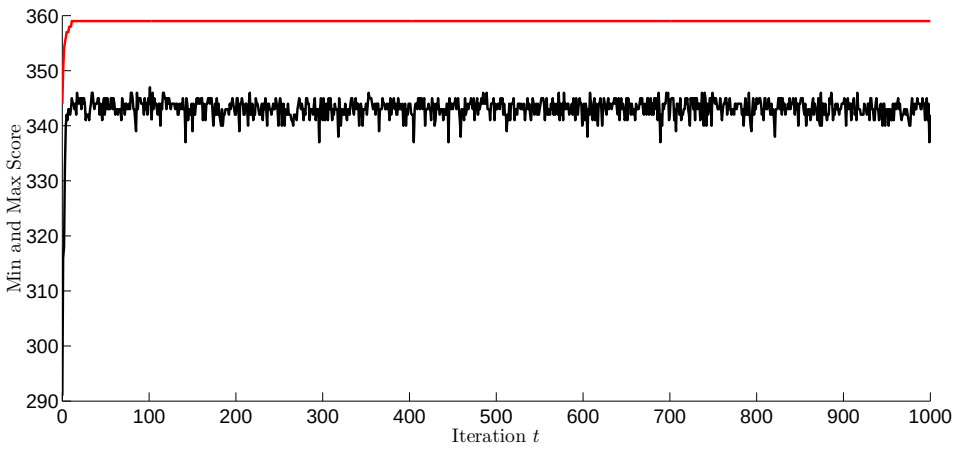


Figure 8.5: Typical best and worst performances of individuals using Algorithm 8.5 on the `F34-5-23-31` problem.

The MATLAB code used for this example can be found on the Handbook website as `GA_ex_fig.m`.

8.4.2 Differential Evolution

The method of **differential evolution** by Price et al. is traditionally applied to continuous optimization problems. In its simplest version, on each iteration,

a new population of points is constructed from the old parent population by moving each of the old points by a fixed step size in a direction determined by taking the difference of two other randomly determined points. The new population then produces a child population through crossover with the old parent population. Finally, each child only replaces its corresponding parent in the new parent population if it has a better performance.

A typical differential evolution algorithm for minimization of a function $S(\mathbf{x})$ is as follows.

Algorithm 8.6 (Differential Evolution Algorithm for Minimization)

1. *Set $t = 0$. Initialize a population of individuals $\mathscr{P}_t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_N^t\}$, say via uniform sampling over a known bounding box.*
2. *For each individual in the population, say individual $\mathbf{x}_k^t$:*
   (a) *Construct a vector $\mathbf{y}_k^{t+1} = \mathbf{x}_{R_1}^t + \alpha\,(\mathbf{x}_{R_2}^t - \mathbf{x}_{R_3}^t)$, where $R_1 \neq R_2 \neq R_3$ are three integers uniformly sampled from the set $\{1, 2, \ldots, k-1, k+1, \ldots, N\}$.*
   (b) *Apply* **binary crossover** *between $\mathbf{y}_k^{t+1}$ and $\mathbf{x}_k^t$ to obtain a trial vector $\widetilde{\mathbf{x}}_k^{t+1}$; that is,*
   $$\widetilde{\mathbf{x}}_k^{t+1} = \left(U_1\, y_{k,1}^{t+1} + (1-U_1)\, x_{k,1}^t, \ldots, U_n\, y_{k,n}^{t+1} + (1-U_n)\, x_{k,n}^t\right),$$
   where $U_1, \ldots, U_d \sim_{\text{iid}} \mathsf{Ber}(p)$. Additionally, select a random index I, uniformly distributed on $\{1, \ldots, n\}$ and set $\widetilde{x}_{k,I}^{t+1} = y_{k,I}^{t+1}$.
   (c) *If $S(\widetilde{\mathbf{x}}_k^{t+1}) \leqslant S(\mathbf{x}_k^t)$, set $\mathbf{x}_k^{t+1} = \widetilde{\mathbf{x}}_k^{t+1}$; otherwise, retain the old individual via $\mathbf{x}_k^{t+1} = \mathbf{x}_k^t$.*
3. *If a stopping criterion is met, stop; otherwise, set $t = t + 1$ and repeat from Step 2.*

The **scaling factor** α and the **crossover factor** p are algorithm parameters. Typical values to try are $\alpha = 0.8$ and $p = 0.9$, with a suggested population size of $N = 10\,n$.

Example 8.5 (Differential Evolution) We illustrate differential evolution by applying it to minimize the 50-dimensional Rosenbrock function, given by

$$S(\mathbf{x}) = \sum_{i=1}^{49} \left(100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right), \tag{8.5}$$

which has minimal value of $S(\mathbf{x}) = 0$ at $\mathbf{x} = (1, \ldots, 1)$.

The population size is $N = 50$, the scaling factor is $\alpha = 0.8$, and the crossover probability is $p = 0.9$. The population is initialized by sampling uniformly on $[-50, 50]^{50}$ and is stopped after 5×10^4 iterations.

Figure 8.6 shows the typical progress of differential evolution on this problem. The best and worst performance function values per iteration are depicted. A MATLAB implementation follows.

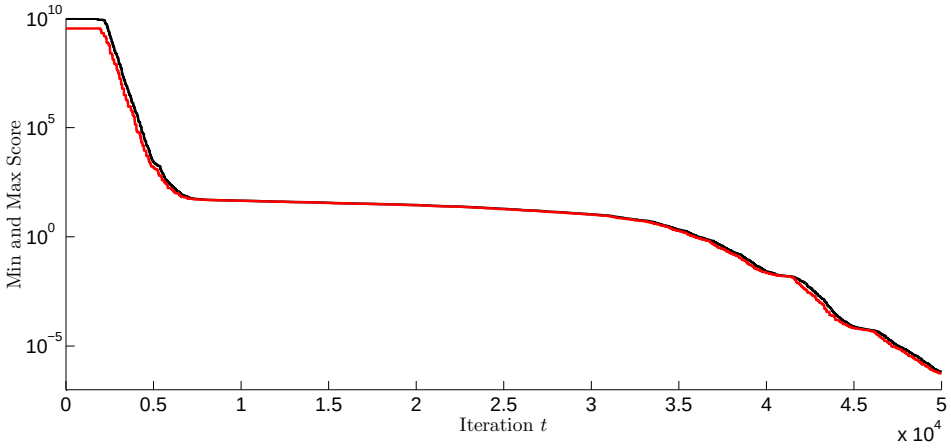


Figure 8.6: Best and worst performance function values on a typical run of a differential evolution algorithm on the 50-dimensional Rosenbrock function.

```
%DE_ex.m
M=50; % Population Size
n=50; % Dimension of the problem
F=0.8; % Scaling factor
CR=0.9; % Crossover factor
maxits=5*10^4; % Maximum number of iterations
Smaxhist=NaN.*ones(1,maxits); Sminhist=NaN.*ones(1,maxits);
% Rosenbrock Function
S=@(X)sum(100.*(X(:,2:n)-X(:,1:(n-1)).^2).^2+ ...
          (X(:,1:(n-1))-1).^2,2);
a=-50; b=50; X=(b-a).*rand(M,n)+a; % Initialize population
t=1; % Iteration Counter
while (t<maxits)
    SX=S(X); [SX,idx]=sort(SX,1,'ascend'); % Score and Sort
    Smaxhist(t)=SX(M); Sminhist(t)=SX(1); % Update histories
    % Construct the new generation
    for i=1:M
        % Mutation
        r=[1:i-1,i+1:M];
        r=r(randperm(M-1));
        V=X(r(1),:)+F.*(X(r(2),:)-X(r(3),:));
        % Binomial Crossover
        U=X(i,:);
        idxr=1+floor(rand(1).*n);
        for j=1:n
            if (rand(1)<=CR)||(j==idxr)
                U(j)=V(j);
            end
        end
```

```
        if S(U)<=S(X(i,:))
            X(i,:)=U;
        end
    end
    t=t+1;
end
SX=S(X); [SX,idx]=sort(SX,1,'ascend'); % Score and Sort
Smaxhist(t)=SX(M); Sminhist(t)=SX(1); % Update histories
% Display worst & best score, and best performing sample
[SX(M),SX(1),X(idx(1),:)]
% Plot the results
figure, plot((1:1:t),Smaxhist,'k-',(1:1:t),Sminhist,'r-')
```

8.5 Exercises

1. Write a simulated annealing algorithm based on the random walk sampler to maximize the function

$$S(x) = \left| \frac{\sin^8(10x) + \cos^5(5x+1)}{x^2 - x + 1} \right|, \quad x \in \mathbb{R} .$$

Use a $\mathsf{N}(x, \sigma^2)$ proposal function, given the current state x. Start with $x = 0$. Plot the current best function value against the number of evaluations of S for various values of σ and various annealing schedules. Repeat the experiments several times to assess what works best.

2. Implement a stochastic counterpart algorithm for the noisy optimization problem in Example 8.1, and compare it with the stochastic approximation method in that example.

Chapter 9

Cross-Entropy Method

The cross-entropy methodology provides a systematic way to design simple and efficient simulation procedures. This chapter describes the method for:

1. *Importance sampling* (see also Section 6.5.3); ☞ 117
2. *Rare-event simulation*;
3. *Optimization*, with examples of *discrete*, *continuous*, and *noisy* problems.

9.1 Cross-Entropy Method

The **cross-entropy (CE) method** is a generic Monte Carlo technique for solving complicated estimation and optimization problems. The approach was introduced by R.Y. Rubinstein, extending his earlier work on variance minimization methods for rare-event probability estimation.

The CE method can be applied to two types of problems:

1. **Estimation**: Estimate $\ell = \mathbb{E}H(\mathbf{X})$, where $\mathbf{X}$ is a random variable or vector taking values in some set $\mathscr{X}$ and H is a function on $\mathscr{X}$. An important special case is the estimation of a probability $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$, where S is another function on $\mathscr{X}$.
2. **Optimization**: Maximize or minimize $S(\mathbf{x})$ over all $\mathbf{x} \in \mathscr{X}$, where S is an objective function on $\mathscr{X}$. S can be either a known or a noisy function. In the latter case the objective function needs to be estimated, for example, via simulation.

In the estimation setting of Section 9.2, the CE method can be viewed as an adaptive importance sampling procedure that uses the CE or Kullback–Leibler divergence as a measure of closeness between two sampling distributions. In the optimization setting of Section 9.3, the optimization problem is first translated into a rare-event estimation problem and then the CE method for estimation is used as an adaptive algorithm to locate the optimum.

9.2 Cross-Entropy Method for Estimation

Consider the estimation of

$$\ell = \mathbb{E}_f H(\mathbf{X}) = \int H(\mathbf{x})\, f(\mathbf{x})\, \mathrm{d}\mathbf{x}\,, \tag{9.1}$$

where H is a sample performance function and f is the probability density of the random vector $\mathbf{X}$. For notational convenience it is assumed that $\mathbf{X}$ is continuous. If $\mathbf{X}$ is discrete, simply replace the integral in (9.1) by a sum. Let g be another probability density such that $g(\mathbf{x}) = 0$ implies that $H(\mathbf{x})\, f(\mathbf{x}) = 0$ for all $\mathbf{x}$. Then, we can represent ℓ as

$$\ell = \int H(\mathbf{x})\, \frac{f(\mathbf{x})}{g(\mathbf{x})}\, g(\mathbf{x})\, \mathrm{d}\mathbf{x} = \mathbb{E}_g H(\mathbf{X})\, \frac{f(\mathbf{X})}{g(\mathbf{X})}\,. \tag{9.2}$$

Consequently, if $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} g$, then

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k)\, \frac{f(\mathbf{X}_k)}{g(\mathbf{X}_k)} \tag{9.3}$$

☞ 113

is an unbiased *importance sampling* estimator of ℓ. The optimal (minimum variance) importance sampling probability density is given by

$$g^*(\mathbf{x}) \propto |H(\mathbf{x})|\, f(\mathbf{x})\,, \tag{9.4}$$

whose normalization constant is unknown. The idea of the CE method is to choose the importance sampling density g in a specified class of densities $\mathscr{G}$ such that the *Kullback–Leibler divergence* (see (6.23) on Page 117) between the optimal importance sampling density g^* and g is minimal. That is, find a $g \in \mathscr{G}$ that minimizes

$$\mathcal{D}(g^*, g) = \mathbb{E}_{g^*} \left[\ln \frac{g^*(\mathbf{X})}{g(\mathbf{X})} \right]. \tag{9.5}$$

In most cases of interest the sample performance function H is non-negative, and the nominal probability density f is parameterized by a finite-dimensional vector $\mathbf{u}$; that is, $f(\mathbf{x}) = f(\mathbf{x}; \mathbf{u})$. It is then customary to choose the importance sampling probability density g in the *same* family of probability densities; thus, $g(\mathbf{x}) = f(\mathbf{x}; \mathbf{v})$ for some **reference parameter** $\mathbf{v}$. The CE minimization procedure then reduces to finding an optimal reference parameter vector, $\mathbf{v}^*$ say, by cross-entropy minimization:

$$\mathbf{v}^* = \operatorname*{argmax}_{\mathbf{v}} \int H(\mathbf{x}) f(\mathbf{x}; \mathbf{u}) \ln f(\mathbf{x}; \mathbf{v})\, \mathrm{d}\mathbf{x}\,, \tag{9.6}$$

which in turn can be estimated via simulation by solving with respect to $\mathbf{v}$ the

☞ 142

stochastic counterpart program

$$\max_{\mathbf{v}} \frac{1}{N} \sum_{k=1}^{N} H(\mathbf{X}_k) \frac{f(\mathbf{X}_k; \mathbf{u})}{f(\mathbf{X}_k; \mathbf{w})}\, \ln f(\mathbf{X}_k; \mathbf{v})\,, \tag{9.7}$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} f(\cdot; \mathbf{w})$, for any reference parameter $\mathbf{w}$. The maximization (9.7) can often be solved *analytically*, in particular when the class of sampling distributions forms an exponential family. Analytical formulas for the solution to (9.7) can be found whenever explicit expressions for the maximum likelihood estimators of the parameters can be found.

Often ℓ in (9.1) is of the form $\mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$ for some performance function S and level γ, in which case $H(\mathbf{x})$ takes the form of an indicator function: $H(\mathbf{x}) = \mathrm{I}_{\{S(\mathbf{X}) \geqslant \gamma\}}$, so that (9.7) becomes

$$\max_{\mathbf{v}} \frac{1}{N} \sum_{\mathbf{X}_k \in \mathscr{E}} \frac{f(\mathbf{X}_k; \mathbf{u})}{f(\mathbf{X}_k; \mathbf{w})} \ln f(\mathbf{X}_k; \mathbf{v}) \,, \tag{9.8}$$

and $\mathscr{E}$ is the **elite** set of samples: those $\mathbf{X}_k$ for which $S(\mathbf{X}_k) \geqslant \gamma$.

A complication in solving (9.8) occurs when ℓ is a rare-event probability; that is, a very small probability (say less than 10^{-4}). Then, for a moderate sample size N most or all of the values $H(\mathbf{X}_k)$ in (9.8) are zero, and the maximization problem becomes useless. To overcome this difficulty, the following *multilevel* CE procedure is used.

Algorithm 9.1 (Multilevel CE Algorithm for Estimating $\mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$)

1. *Define* $\widehat{\mathbf{v}}_0 = \mathbf{u}$. *Let* $N^{\mathrm{e}} = \lceil \varrho N \rceil$. *Set* $t = 1$ *(iteration counter).*

2. *Generate* $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\text{iid}} f(\cdot; \widehat{\mathbf{v}}_{t-1})$. *Calculate the performances* $S(\mathbf{X}_i)$ *for all* i *and order them from smallest to largest:* $S_{(1)} \leqslant \ldots \leqslant S_{(N)}$. *Let* $\widehat{\gamma}_t$ *be the sample* $(1-\varrho)$*-quantile of performances; that is,* $\widehat{\gamma}_t = S_{(N-N^{\mathrm{e}}+1)}$. *If* $\widehat{\gamma}_t > \gamma$, *reset* $\widehat{\gamma}_t$ *to* γ.

3. *Use the* same *sample* $\mathbf{X}_1, \ldots, \mathbf{X}_N$ *to solve the stochastic program* (9.8) *with* $\mathbf{w} = \widehat{\mathbf{v}}_{t-1}$. *Denote the solution by* $\widehat{\mathbf{v}}_t$.

4. *If* $\widehat{\gamma}_t < \gamma$, *set the counter* $t = t + 1$ *and reiterate from Step 2; otherwise, proceed with Step 5.*

5. *Let* T *be the final iteration counter. Generate* $\mathbf{X}_1, \ldots, \mathbf{X}_{N_1} \sim_{\text{iid}} f(\cdot; \widehat{\mathbf{v}}_T)$ *and estimate* ℓ *via importance sampling as in* (9.3).

The algorithm requires specification of the family of sampling probability densities $\{f(\cdot; \mathbf{v}), \mathbf{v} \in \mathscr{V}\}$, the sample sizes N and N_1, and the rarity parameter ϱ (typically between 0.01 and 0.1). Typical values for the sample sizes are $N = 10^3$ and $N_1 = 10^5$. Under certain technical conditions the deterministic version of Algorithm 9.1 is guaranteed to terminate (reach level γ) provided that ϱ is chosen small enough.

Example 9.1 (Rare-Event Probability Estimation) Suppose the problem is to estimate $\ell = \mathbb{P}(\min\{X_1, \ldots, X_n\} \geqslant \gamma)$, where $X_k \sim \mathsf{Beta}(1, u_k/(1-u_k))$, $k = 1, \ldots, n$ independently. Note that this parameterization ensures that

$\mathbb{E}X_k = u_k$, and that we do not assume that the $\{u_k\}$ are the same. However, if $u_1 = \cdots = u_n = 1/2$, then we have $X_1, \ldots, X_n \sim_{\text{iid}} \mathsf{U}(0,1)$. In that case $\ell = (1-\gamma)^n$. In particular, if we take $n = 5$, $\gamma = 0.999$, and $u_k = 1/2$ for all k, then $\ell = 10^{-15}$.

For these particular problem parameters, typical output of Algorithm 9.1 using a rarity parameter of $\varrho = 0.1$, and sample sizes of $N = 10^3$ and $N_1 = 10^6$, is given below.

Table 9.1: Typical convergence of parameter vectors with multilevel CE.

t	$\widehat{\gamma}_t$	$\widehat{v}_t$				
0	-	0.5	0.5	0.5	0.5	0.5
1	0.60117	0.79938	0.80341	0.79699	0.79992	0.80048
2	0.88164	0.93913	0.94094	0.94190	0.94138	0.94065
3	0.96869	0.98423	0.98429	0.98446	0.98383	0.98432
4	0.99184	0.99586	0.99588	0.99590	0.99601	0.99590
5	0.99791	0.99896	0.99895	0.99893	0.99897	0.99896
6	0.999	0.99950	0.99949	0.99949	0.99950	0.99950

This gives a final estimate of $\widehat{\ell} = 1.0035 \times 10^{-15}$ with an estimated relative error of 0.003.

In this example, we can calculate the exact CE optimal parameters from (9.6). With $u_1 = \cdots = u_n = 1/2$, and due to independence, each component of the optimal parameter vector is solved from

$$v^* = \underset{v \in (0,1)}{\operatorname{argmax}} \int_\gamma^1 \ln\left(\left(\frac{v-1}{v}\right) x^{(v/(v-1)-1)}\right) \mathrm{d}x\,.$$

The solution is given by

$$v^* = \frac{1-\gamma}{2(1-\gamma) + \gamma \ln \gamma}\,.$$

With $\gamma = 0.999$, this gives $v^* = 0.99950$ to five significant digits, which is as found via the multilevel algorithm in Table 9.1. MATLAB code for this example follows.

```
%CEest.m
f='minbeta'; % performance function name
gam=0.999; % Desired level parameter
n=5;  % dimension of problem
N=10^5; % sample size
rho=0.01;
N1=10^6; % Final estimate sample size
N_el=round(N*rho);  % elite sample size
u=0.5.*ones(1,n); % Nominal ref. parameter in Beta(1,u/(1-u))
v=u; gt=-inf; % Initialize v and gamma
```

```
maxits=10^5; % Fail-safe stopping after maxits exceeded
it=0; tic
while (gt<gam)&(it<maxits)
  it=it+1;
  % Generate and score X's
  X=rand(N,n).^(1./repmat(v./(1-v),N,1)); % Beta(1,v/(1-v))
  S=feval(f,X); [U,I]=sort(S);  % Score & Sort
  % Update Gamma_t
  gt=U(N-N_el+1);
  if gt>gam, gt=gam; N_el=N-find(U>=gam,1)+1; end
  Y=X(I(N-N_el+1:N),:);
  % Calculate likelihood ratios and update v
  W=prod(repmat((u./(1-u))./(v./(1-v)),N_el,1).*...
      Y.^(repmat((u./(1-u))-(v./(1-v)),N_el,1)),2);
  v=sum(repmat(W,1,n).*Y)./sum(repmat(W,1,n));
  [gt,v] % Display gamma and v
end
% Final estimation step
X1=rand(N1,n).^(1./repmat(v./(1-v),N1,1));
S1=feval(f,X1);
W1=prod(repmat((u./(1-u))./(v./(1-v)),N1,1).*...
    X1.^(repmat((u./(1-u))-(v./(1-v)),N1,1)),2);
H1=(S1>=gam);
ell=mean(W1.*H1);
re=sqrt((mean((W1.*H1).^2)/(ell^2))-1)/sqrt(N1);
% Display final results
time=toc; disp(time), disp(v), disp(ell), disp(re)
ell_true=(1-gam)^n;disp(ell_true) % Display true quantity
```

```
function out=minbeta(X)
out=min(X,[],2);
```

9.3 Cross-Entropy Method for Optimization

Let S be a real-valued performance function on $\mathscr{X}$. Suppose we wish to find the maximum of S over $\mathscr{X}$, and the corresponding state $\mathbf{x}^*$ at which this maximum is attained (assuming for simplicity that there is only one such state). Denoting the maximum by γ^*, we thus have

$$S(\mathbf{x}^*) = \gamma^* = \max_{\mathbf{x}\in\mathscr{X}} S(\mathbf{x}) \,. \tag{9.9}$$

This setting includes many types of optimization problems: discrete (combinatorial), continuous, mixed, and constrained problems. If one is interested in minimizing rather than maximizing S, one can simply maximize $-S$.

Now associate with the above problem the estimation of the probability $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$, where $\mathbf{X}$ has some pdf $f(\mathbf{x}; \mathbf{u})$ on $\mathscr{X}$ (for example corresponding to the uniform distribution on $\mathscr{X}$) and γ is some level. If γ is chosen close to the unknown γ^*, then ℓ is typically a rare-event probability, and the CE approach of Section 9.2 can be used to find an importance sampling distribution close to the theoretically optimal importance sampling density, which concentrates all its mass at the point $\mathbf{x}^*$. Sampling from such a distribution thus produces optimal or near-optimal states. A main difference with the CE method for rare-event simulation is that, in the optimization setting, the final level $\gamma = \gamma^*$ is not known in advance. The CE method for optimization produces a sequence of levels $\{\widehat{\gamma}_t\}$ and reference parameters $\{\widehat{\mathbf{v}}_t\}$ such that the former tends to the optimal γ^* and the latter to the optimal reference vector $\mathbf{v}^*$ corresponding to the point mass at $\mathbf{x}^*$.

Given the class of sampling probability densities $\{f(\cdot; \mathbf{v}), \mathbf{v} \in \mathscr{V}\}$, the sample size N, and the rarity parameter ϱ, the CE algorithm for optimization is as follows.

Algorithm 9.2 (CE Algorithm for Optimization)

1. *Choose an initial parameter vector $\widehat{\mathbf{v}}_0$. Let $N^{\mathrm{e}} = \lceil \varrho N \rceil$. Set the level counter to $t = 1$.*

2. *Generate $\mathbf{X}_1, \ldots, \mathbf{X}_N \sim_{\mathrm{iid}} f(\cdot; \widehat{\mathbf{v}}_{t-1})$. Calculate the performances $S(\mathbf{X}_i)$ for all i and order them from smallest to largest: $S_{(1)} \leqslant \ldots \leqslant S_{(N)}$. Let $\widehat{\gamma}_t$ be the sample $(1-\varrho)$-quantile of performances; that is, $\widehat{\gamma}_t = S_{(N-N^{\mathrm{e}}+1)}$.*

3. *Use the* same *sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ to determine the elite set of samples $\mathscr{E}_t = \{\mathbf{X}_k : S(X_k) \geqslant \widehat{\gamma}_t\}$ and solve the stochastic program*

$$\max_{\mathbf{v}} \sum_{\mathbf{X}_k \in \mathscr{E}_t} \ln f(\mathbf{X}_k; \mathbf{v}) \,. \tag{9.10}$$

   Denote the solution by $\widehat{\mathbf{v}}_t$.

4. *If some stopping criterion is met, stop; otherwise, set $t = t+1$, and return to Step 2.*

Any CE algorithm for optimization thus involves the following two main iterative phases:

1. **Generate** an iid sample of objects in the search space $\mathscr{X}$ (trajectories, vectors, etc.) according to a specified probability distribution.

2. **Update** the parameters of that distribution, based on the N^{e} best performing samples (the elite samples), using cross-entropy minimization.

There are two key differences between Algorithm 9.1 and Algorithm 9.2: (1) Step 5 is missing for optimization, and (2) the likelihood ratio term $f(\mathbf{X}_k; \mathbf{u})/f(\mathbf{X}_k; \widehat{\mathbf{v}}_{t-1})$ in (9.7) is missing in (9.10).

Often a smoothed updating rule is used, in which the parameter vector $\widehat{\mathbf{v}}_t$ is taken as

$$\widehat{\mathbf{v}}_t = \operatorname{diag}(\boldsymbol{\alpha})\,\widetilde{\mathbf{v}}_t + \operatorname{diag}(\mathbf{1}-\boldsymbol{\alpha})\,\widehat{\mathbf{v}}_{t-1}\,, \tag{9.11}$$

where $\widetilde{\mathbf{v}}_t$ is the solution to (9.10) and $\boldsymbol{\alpha}$ is a vector of **smoothing parameters**, with each component in $[0,1]$. Many other modifications exist. When there are two or more optimal solutions, the CE algorithm typically "fluctuates" between the solutions before focusing on one of the solutions.

9.3.1 Combinatorial Optimization

When the state space $\mathscr{X}$ is finite, the optimization problem (9.9) is often referred to as a **discrete** or **combinatorial optimization** problem. For example, $\mathscr{X}$ could be the space of combinatorial objects such as binary vectors, trees, paths through graphs, etc. To apply the CE method, one first needs to specify a convenient parameterized random mechanism to generate objects in $\mathscr{X}$. For example, when $\mathscr{X}$ is the set of binary vectors of length n, an easy generation mechanism is to draw each component independently from a Bernoulli distribution; that is, $\mathbf{X} = (X_1,\ldots,X_n)$, where $X_i \sim \mathsf{Ber}(p_i)$, $i = 1,\ldots,n$, independently. Given an elite sample set $\mathscr{E}$ of size N^{e}, the updating formula is then

$$\widehat{p}_i = \frac{\sum_{\mathbf{X}\in\mathscr{E}} X_i}{N^{\mathrm{e}}}, \quad i = 1,\ldots,n\,. \tag{9.12}$$

A possible stopping rule for combinatorial optimization problems is to stop when the overall best objective value does not change over a number of iterations. Alternatively, one could stop when the sampling distribution has "degenerated" enough. In particular, in the Bernoulli case (9.12) one could stop when all $\{p_i\}$ are less than some small distance $\varepsilon > 0$ away from either 0 or 1.

Example 9.2 (Satisfiability Problem) We illustrate the CE optimization Algorithm 9.2 by applying it to the satisfiability problem (SAT) considered in Example 8.4. ☞ 150

We take our sampling pdf g to be of the form

$$g(\mathbf{x}) = \prod_{i=1}^{n} p_i^{x_i}(1-p_i)^{1-x_i}.$$

In this case, the i-th component of $\mathbf{x}$ is generated according to the $\mathsf{Ber}(p_i)$ distribution, independently of all other components.

The Bernoulli probabilities are updated using (9.12), where the set of elite samples $\mathscr{E}$ on iteration t is the proportion ϱ of best performers — in this case, the proportion ϱ of samples that satisfy the most clauses.

If we write $\widehat{\mathbf{p}}_t = (\widehat{p}_{t1},\ldots,\widehat{p}_{tn})$ for the solution from (9.12) in iteration t, then the idea is that the sequence $\widehat{\mathbf{p}}_0, \widehat{\mathbf{p}}_1, \ldots$ converges to one of the solutions to the satisfiability problem.

We run the algorithm with a sample size of $N = 10^4$ and a rarity parameter of $\varrho = 0.1$, giving $N^{\mathrm{e}} = 10^3$ elite samples per iteration. We take

$\widehat{\mathbf{p}}_0 = (0.5, \ldots, 0.5)$ and use a constant smoothing parameter of $\alpha = 0.5$. Finally, our algorithm is stopped after 10^3 iterations, or if the vector $\mathbf{p}_t$ has *degenerated*: if every component p_{tk} is within $\varepsilon = 10^{-3}$ of either 0 or 1.

In Figure 9.1, the scores of the best and worst performing elite samples are plotted for the problem `F34-5-23-31`. As with the binary coded genetic algorithm in Example 8.4, the best solutions found only satisfied 359 out of 361 clauses.

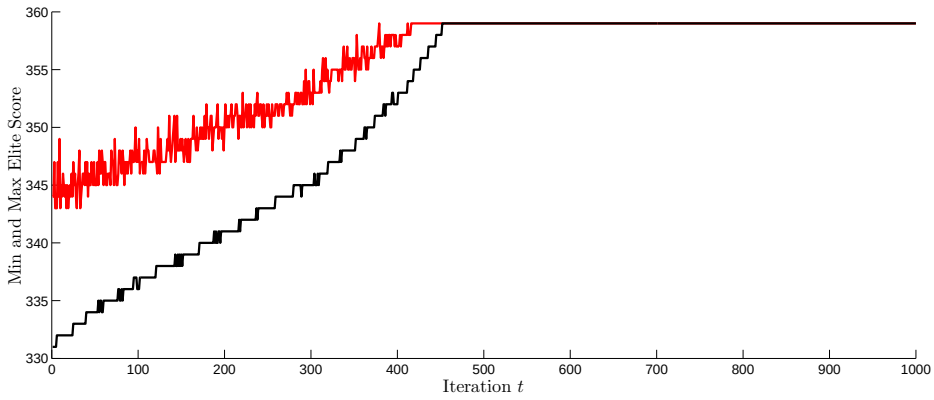


Figure 9.1: Typical best and worst elite samples using Algorithm 9.2 on the `F34-5- 23-31` problem.

MATLAB code that implements the basic algorithm is given below. It is assumed there is an efficient function `sC.m` that accepts N trial vectors and computes the number of clauses that each satisfies. Our implementation of `sC.m` is available on the Handbook website.

```
%CESAT.m
%Assumes *sparse* A is loaded in the workspace
[m,n]=size(A); % Dimensions of the problem
maxits=10^3; epsilon=1e-3;
N=10^4; rho=0.1; Ne=ceil(rho*N);
alpha=0.7; % Smoothing parameter
p = 0.5*ones(1,n); it=0; best=-inf; xbest=[];
Smaxhist=zeros(1,maxits); % Allocate history memory
Sminhist=zeros(1,maxits);
while (max(min(p,1-p)) > epsilon) && (it < maxits) && (best<m)
  it = it + 1;
  x = double((rand(N,n) < repmat(p,N,1)));
  SX = sC(A,x);
  [sortSX,iX] = sortrows([x SX],n+1);
  indices=iX(N- Ne + 1:N);
  if sortSX(N,n+1)>best
    best=sortSX(N,n+1); xbest=sortSX(N,1:n);
  end
```

```
  Smaxhist(it)=sortSX(N,n+1);Sminhist(it)=sortSX(N-Ne+1,n+1);
  p=alpha.*mean(sortSX(indices,1:n))+(1-alpha).*p;
  disp([it,sortSX(N,n+1),sortSX(N-Ne+1,n+1),p])
end
disp([best xbest])
figure,plot((1:1:it),Smaxhist,'r-',(1:1:it),Sminhist,'k-')
```

9.3.2 Continuous Optimization

It is also possible to apply the CE algorithm to continuous optimization problems; in particular, when $\mathscr{X} = \mathbb{R}^n$. The sampling distribution on $\mathbb{R}^n$ can be quite arbitrary and does not need to be related to the function that is being optimized. The generation of a random vector $\mathbf{X} = (X_1, \ldots, X_n) \in \mathbb{R}^n$ is usually established by drawing the coordinates independently from some two-parameter distribution. In most applications a normal (Gaussian) distribution is employed for each component. Thus, the sampling distribution for $\mathbf{X}$ is characterized by a vector $\boldsymbol{\mu}$ of means and a vector $\boldsymbol{\sigma}$ of standard deviations. At each iteration of the CE algorithm these parameter vectors are updated simply as the vectors of sample means and sample standard deviations of the elements in the elite set. During the course of the algorithm, the sequence of mean vectors ideally tends to the maximizer $\mathbf{x}^*$, while the vector of standard deviations tend to the zero vector. In short, one should obtain a degenerated probability density with all mass concentrated in the vicinity of the point $\mathbf{x}^*$. A possible stopping criterion is to stop when all standard deviations are smaller than some ε. This scheme is referred to as **normal updating**.

In what follows, we give examples of CE applied to unconstrained, constrained, and noisy continuous optimization problems. In each case, we employ normal updating.

Example 9.3 (Maximizing the Peaks Function) Suppose we wish to maximize MATLAB's peaks function, given by

$$S(\mathbf{x}) = 3\,(1-x_1)^2\,\mathrm{e}^{-x_1^2-(x_2+1)^2} - 10\left(\frac{x_1}{5} - x_1^3 - x_2^5\right)\mathrm{e}^{-x_1^2-x_2^2} - \frac{1}{3}\,\mathrm{e}^{-(x_1+1)^2-x_2^2}\,.$$

This function has three local maxima and three local minima, with global maximum at $\mathbf{x}^* \approx (-0.0093, 1.58)$ of $\gamma^* = S(\mathbf{x}^*) \approx 8.1$.

With normal updating, the choice of the initial value for $\boldsymbol{\mu}$ is not important, so we choose $\boldsymbol{\mu} = (-3, -3)$ arbitrarily. However, the initial standard deviations should be chosen large enough to ensure an initially "uniform" sampling of the region of interest, hence $\boldsymbol{\sigma} = (10, 10)$ is chosen. The CE algorithm is stopped when all standard deviations of the sampling distribution are less than some small ε, say $\varepsilon = 10^{-5}$. A typical evolution of the mean of the sampling distribution is depicted in Figure 9.2.

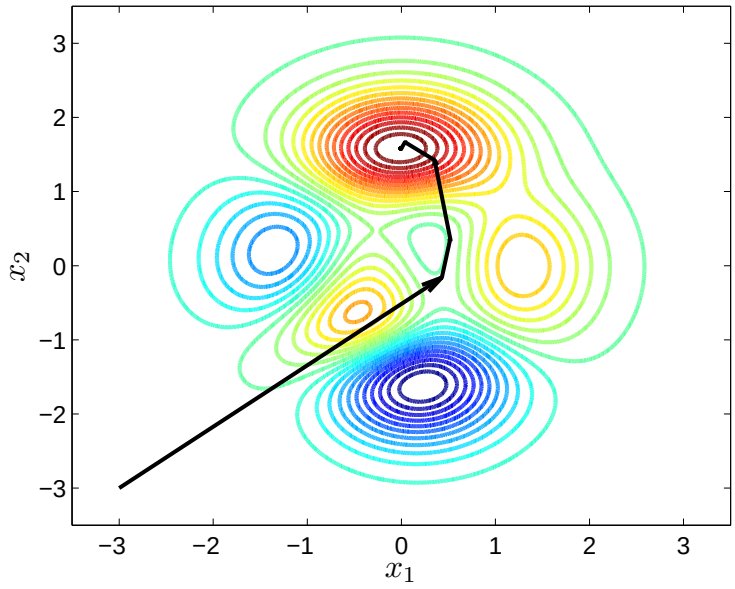


Figure 9.2: A typical evolution of the mean vector with normal updating for the peaks function.

A MATLAB implementation of CE Algorithm 9.2 is given below. The peaks function is stored in a separate file `S.m`.

```
%peaks\simplepeaks.m
n = 2;                                  % dimension
mu = [-3,-3]; sigma = 3*ones(1,n); N = 100; eps = 1E-5; rho=0.1;
while max(sigma) > eps
   X = randn(N,n)*diag(sigma)+ mu(ones(N,1),:);
   SX= S(X);                            %Compute the performance
   sortSX = sortrows([X, SX],n+1);
   Elite = sortSX((1-rho)*N:N,1:n); % elite samples
   mu = mean(Elite,1);                  % sample mean row-wise
   sigma = std(Elite,1);                % sample st.dev. row-wise
   [S(mu),mu,max(sigma)]                % output the result
end
```

```
function out = S(X)
out =  3*(1-X(:,1)).^2.*exp(-X(:,1).^2 - (X(:,2)+1).^2) ...
 - 10*(X(:,1)/5 - X(:,1).^3 - X(:,2).^5) ...
 .*exp(-X(:,1).^2-X(:,2).^2)-1/3*exp(-(X(:,1)+1).^2-X(:,2).^2);
```

9.3.3 Constrained Optimization

In order to apply the CE method to constrained maximization problems, we must first put the problem in the framework of (9.9). Let $\mathscr{X}$ be a region defined by some system of inequalities:

$$G_i(\mathbf{x}) \leqslant 0, \quad i = 1, \ldots, k \,. \tag{9.13}$$

Two approaches can be adopted to solve the program (9.9) with constraints (9.13). The first approach uses *acceptance–rejection*: generate a random vector $\mathbf{X}$ from, for example, a multivariate normal distribution with independent components, and accept or reject it depending on whether the sample falls in $\mathscr{X}$ or not. Alternatively, one could sample directly from a truncated distribution (for example, a truncated normal distribution) or combine such a method with acceptance-rejection. Once a fixed number of such vectors has been accepted, the parameters of the normal distribution can be updated in exactly the same way as in the unconstrained case — simply via the sample mean and standard deviation of the elite samples. A drawback of this method is that a large number of samples could be rejected before a feasible sample is found.

The second approach is the *penalty approach*. Here, the objective function is modified to

$$\widetilde{S}(\mathbf{x}) = S(\mathbf{x}) + \sum_{i=1}^{k} H_i \, \max\{G_i(\mathbf{x}), 0\} \,, \tag{9.14}$$

where $H_i < 0$ measures the importance (cost) of the i-th penalty.

Thus, by reducing the constrained problem ((9.9) and (9.13)) to an unconstrained one ((9.9) with $\widetilde{S}$ instead of S), one can again apply Algorithm 9.2.

Example 9.4 (MLE for the Dirichlet Distribution) Suppose that we are given data $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim_{\text{iid}} \mathsf{Dirichlet}(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^\top$ is an unknown parameter vector satisfying $\alpha_i > 0$, $i = 1, \ldots, K$. The conditions on $\boldsymbol{\alpha}$ provide natural inequality constraints: $G_i(\boldsymbol{\alpha}) \equiv -\alpha_i \leqslant 0$, $i = 1, \ldots, K$.

We will use Algorithm 9.2 with normal updating to obtain the MLE by direct maximization of the log-likelihood for the Dirichlet distribution given the data. However, due to the constraints for a valid parameter vector $\boldsymbol{\alpha}$, we apply a penalty of $H_1 = \cdots = H_K = -\infty$ whenever a constraint is violated.

Figure 9.3 displays the distance between the mean vector of the sampling distribution and the true MLE calculated via a fixed-point technique for a data size of $n = 100$ points from the $\mathsf{Dirichlet}(1, 2, 3, 4, 5)$ distribution. The CE parameters are a sample size of $N = 10^4$ and an elite sample size of $N^{\mathrm{e}} = 10^3$. No smoothing parameter is applied to the mean vector, but a constant smoothing parameter of 0.5 is applied to each of the standard deviations.

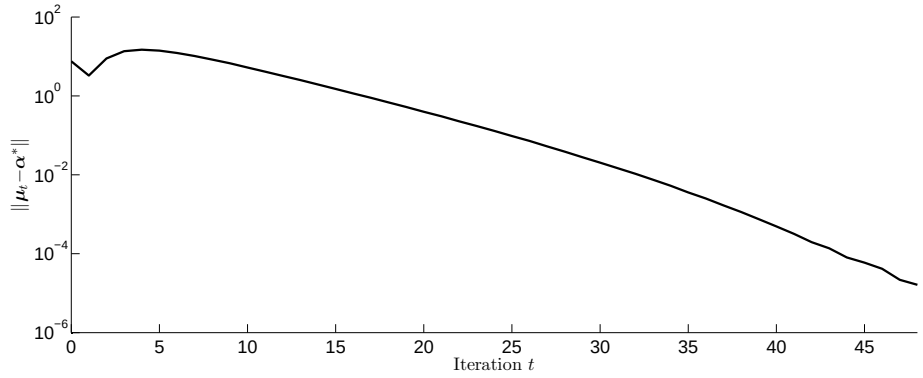


Figure 9.3: Typical evolution of the Euclidean distance between the mean vector and the MLE $\boldsymbol{\alpha}^*$.

```
%MLE\ce_dirichlet_mle_fp.m
clear all;
a=(1:1:5); n=100;
K=length(a); %K dim vector
data=dirichletrnd(a,n,K); % Generate data
epsilon=10^(-4); % For
N=10^4; rho=0.1; alphamu=1; alphasig=0.5; Ne=ceil(N.*rho);
mu=zeros(1,K); sig=ones(1,K).*10; % Initial parameters
muhist=mu;sighist=sig; % History
while max(sig)>epsilon
    x=repmat(mu,N,1)+repmat(sig,N,1).*randn(N,K); % Sample
    S=dirichlet_log_like(data,x,n,K); [S,I]=sort(S); %Score&Sort
    mu=alphamu.*mean(x(I(N-Ne+1:N),:))+(1-alphamu).*mu;
    sig=alphasig.*std(x(I(N-Ne+1:N),:),1,1)+(1-alphasig).*sig;
    muhist=[muhist;mu];sighist=[sighist;sig]; % Update History
    [mu, sig, S(end),S(N-Ne+1)]
end
% For comparison, compute the MLE using a fixed-point method
afp=dirichlet_MLE_FP(data,K);
disp([afp,dirichlet_log_like(data,afp,n,K)])
```

The function `dirichlet_log_like.m` calculates the log-likelihood of the set of trial parameters for the given data set.

```
function out=dirichlet_log_like(data,x,n,K)
out=zeros(size(x,1),1);
I=any(x<=0,2);nI=~I;
out(I)=-inf;
out(nI)=n.*(log(gamma(sum(x(nI,:),2)))-sum(log(gamma(x(nI,:))),2));
```

```
for k=1:n
    out(nI)=out(nI)+sum((x(nI,1:(K-1))-1).*...
      repmat(log(data(k,1:(K-1))),sum(nI),1),2)+(x(nI,K)-1).*...
      repmat(log(1-sum(data(k,1:(K-1)),2)),sum(nI),1);
end
```

The function `dirichletrnd.m` generates Dirichlet distributed realizations as in Algorithm 3.15. ☞ 55

```
function out=dirichletrnd(a,n,K)
out=zeros(n,K);
for k=1:n
  temp=zeros(1,K);
  for i=1:(K)
    temp(i)=gamrnd(a(i),1);
  end
  out(k,:)=temp./sum(temp);
end
```

The function `dirichlet_MLE_FP.m` computes the MLE using a fixed-point technique.

```
function afp=dirichlet_MLE_FP(data,K)
%Compute Dirichlet MLE via a fixed-point technique
logpdata=mean(log(data),1);
afp=ones(1,K); afpold=-inf.*afp;
while sqrt(sum((afp-afpold).^2))>10^(-12)
   afpold=afp; s=sum(afpold);
   for k=1:K
     y=(psi(s)+logpdata(k));
     if y>=-2.22
         ak=exp(y)+0.5;
     else
         ak=-1/(y-psi(1));
     end
     akold=-inf;
     while abs(ak-akold)>10^(-12)
        akold=ak; ak=akold - ((psi(akold)-y)/psi(1,akold));
     end
     afp(k)=ak;
   end
end
```

9.3.4 Noisy Optimization

Noisy (or *stochastic*) optimization problems — in which the objective function is corrupted with noise — arise in many contexts, for example, in stochastic scheduling and stochastic shortest/longest path problems, and simulation-based optimization. The CE method can be easily modified to deal with noisy optimization problems. Consider the maximization problem (9.9) and assume that the performance function is noisy. In particular, suppose that $S(\mathbf{x}) = \mathbb{E}\widehat{S}(\mathbf{x})$ is *not* available, but that a sample value $\widehat{S}(\mathbf{x})$ (unbiased estimate of $\mathbb{E}\widehat{S}(\mathbf{x})$) is available, for example via simulation. The principal modification of the Algorithm 9.2 is to replace $S(\mathbf{x})$ by $\widehat{S}(\mathbf{x})$. In addition, one may need to increase the sample size in order to reduce the effect of the noise. Although various applications indicate the usefulness of the CE approach for noisy optimization, little is known regarding theoretical convergence results in the noisy case. A possible stopping criterion is to stop when the sampling distribution has degenerated enough. Another possibility is to stop the stochastic process when the sequence of levels $\{\widehat{\gamma}_t\}$ has reached stationarity.

Example 9.5 (Noisy Peaks Function) This example is a noisy version of Example 9.3, for which the performance function S has been corrupted by standard normal noise: $\widehat{S}(\mathbf{x}) = S(\mathbf{x}) + \varepsilon$, $\varepsilon \sim \mathsf{N}(0,1)$. The following MATLAB code provides a simple implementation of the CE algorithm to maximize the peaks function when the sample performance values are corrupted by noise in this way. The CE parameters and the function `S.m` are the same as in Example 9.3. Typical evolution of the mean of the sampling distribution is depicted in the Figure 9.4.

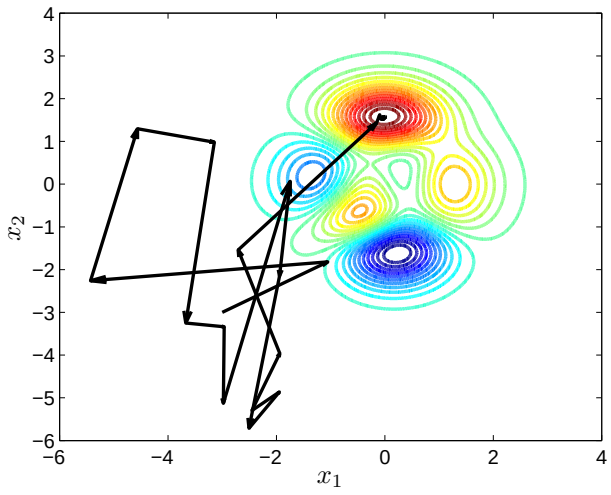


Figure 9.4: Typical evolution of the mean vector with normal updating for the noisy peaks function.

```
%peaks\simplepeaks_noisy.m
n = 2;                                % dimension
mu = [-3,-3]; sigma = 3*ones(1,n); N = 100; eps = 1E-5; rho=0.1;
while max(sigma) > eps
   X = randn(N,n)*diag(sigma)+ mu(ones(N,1),:);
   SX= S(X);                          %Compute the performance
   SX= SX+randn(N,1);                 %Corrupt with noise
   sortSX = sortrows([X, SX],n+1);
   Elite = sortSX((1-rho)*N:N,1:n); % elite samples
   mu = mean(Elite,1);                % sample mean row-wise
   sigma = std(Elite,1);              % sample st.dev. row-wise
   [S(mu)+randn,mu,max(sigma)]             % output the result
end
```

9.4 Exercises

1. Implement a cross-entropy program to maximize the same function as in Question 1 in Section 8.5. Compare the results with those obtained via simulated annealing.

2. The CE tutorial paper, which can be found on the CE webpage `www.cemethod.org` starts with two toy examples — one for rare-event estimation and one for combinatorial optimization. Implement the corresponding CE algorithms in MATLAB and verify the results.

3. Implement a CE algorithm for the maxcut problem, as in Example 3.2 of the CE tutorial.

4. Consider the binary knapsack problem:

$$\max_{\mathbf{x}\in\{0,1\}^n} \mathbf{p}^\top \mathbf{x}\,,$$

subject to the constraint

$$\mathbf{w}^\top \mathbf{x} \leqslant c\,,$$

where both $\mathbf{p}$ and $\mathbf{w}$ are vectors of non-negative numbers. The interpretation is that p_i represents the value and w_i the volume of the i-th item that you want to pack in your knapsack. The total volume of the knapsack is $c < \sum w_i$, so you cannot pack every item.

Write a CE program to solve a non-trivial knapsack problem of size $n = 20$ and $n = 40$. For the smaller problem check that your program finds the maximum by enumerating all possibilities.

5. Implement a CE algorithm for the noisy optimization problem in Example 8.1, and compare it with the stochastic approximation method in that example.

Index

absolutely continuous distribution, 49, 55
acceptance probability, 88
acceptance–rejection method, **38**–**165**
  efficiency, **38**, 41
affine
  combination, 57
  transformation, **28**, 29, 56–58
alias method, 35–38
annealing schedule, 145, 147
antithetic
  estimator, 105
  normal — random variables, 107
  pair, 105
  random variable, 105, 126
asymptotic optimality, 120

balance equations
  detailed, 88, 93, 97
  global, 93
Bayes' rule, 94
Bayesian statistics, 33, 89, 100
Bernoulli
  distribution, **45**, 46, 161
  process, **45**, 46
  trial, 45
beta distribution, 33, 34, **49**–**50**, 55, 118
beta function, 49
binary crossover, 151
binary rank test, 18, 23
binomial distribution, **46**
birthday spacings test, 23
Black–Scholes model, 80
Boltzmann distribution, 145
box constraints, 139
Box–Muller method, 34, **53**
bridge network, **103**, 106, 107, 109, 111, 113, 116, 118, 127, 128, 131, 133
Brownian bridge, **78**
Brownian motion, 73, **74**
  standard, 74

Cauchy distribution, 27, **50**, 143
central difference estimator, **125**–**127**, 129, 140
central limit theorem, 53
χ^2 distribution, 19, **51**, 58
χ^2 test, 18–24
Cholesky factorization, 61, 62
collision test, 23
combinatorial optimization, 161
combined generator, 16, 17
combined multiple-recursive generator, 12, 16, 17
common random variables, **126**, 129, 140
composition method, **32**, 33
conditional
  distribution, 30, 32, 40, 57
  expectation, 111
  Monte Carlo, **110**, 111
    estimator, 111
  pdf, 38, 40, 92
confidence interval, 113
constrained optimization, 165
continuous optimization, 163
control variable, 108, **108**, 109, 110
  estimator, 108
  multiple, 110
convex
  hull, 69

cooling factor in simulated annealing, 146
correlation coefficient
multiple, 110
coupon collector's test, 22
covariance
function, **61**
matrix, 28, 56, 57, 110
credible interval, 94
cross-entropy
distance, 117, 155
method, 117–119, 143, **155**
program, 117, 143
crossover factor in differential evolution, 151
crude Monte Carlo, 104, 128
cumulative distribution function (cdf), 26, 27, 32
empirical, 20

derivatives
estimation of, 123–134, 137
differential evolution algorithm, 150
diffusion
coefficient of Brownian motion, 74
matrix, 76
process, 75–78
Dirichlet distribution, **55**, 165
discrete
distribution, 27, 35, 45
uniform distribution, **48**
distribution
arcsine, *see* beta distribution
Bernoulli, **45**, 46
beta, 33, 34, **49–50**, 55, 118
binomial, **46**
Boltzmann, 145
Cauchy, 27, **50**, 143
χ^2, **51**, 58
Dirichlet, **55**, 165
discrete, 27, 35, 45
uniform, **48**
empirical, 20, 35
exponential, 27, **51**
exponential family, 157
extreme value, 59
gamma, 49, **51–53**, 55, 58
Gaussian, *see* normal distribution
geometric, **46–47**
inverse —, **30**
gamma, **30**
Lorentz, *see* Cauchy distribution
mixture, **32**
multinormal, *see* multivariate normal distribution
multivariate Gaussian, *see* multivariate normal distribution
multivariate normal, **56–58**, 58, 61
multivariate Student's t, **58**
negative exponential, *see* exponential distribution
normal, 29, 39, **53–54**, 56–58
Poisson, **47–48**
positive normal, **39**, 54
reciprocal, **30**
standard normal, **53–54**
Student's t, 58, 90
truncated —, **30**
exponential, 31
multivariate normal, 98
normal, 31
uniform, 27, **54**
distributional parameter, **123**, 125, 129
dominated convergence theorem, 124
dominating density, **113**, 116, 143
drawing, *see* resampling
drift
of a Brownian motion, 74

efficiency
of acceptance–rejection method, **38**, 41
elite sample set in cross-entropy, **157**
empirical
cdf, 20
distribution, 20, 35

equidistribution test, **20**
estimator
  antithetic, 105
  central difference, **125–127**, 129
  conditional Monte Carlo, 111
  control variable, 108
  forward difference, **125**
  importance sampling, 113, 156
  maximum likelihood, 157, 165
  score function, 130, 132
Euler's method, **76**, 100
  multidimensional, 77
evolutionary algorithm, 148
expectation
  function, **61**
exponential distribution, 27, **51**
exponential family, 157
extreme value distribution, 59

finite difference method, **125–127**, 137, 139
finite-dimensional distributions, 61
forward difference estimator, **125**
frequency test, **20**

gamma distribution, 49, **51–53**, 55, 58
gap test, **21**
Gaussian distribution, *see* normal distribution
Gaussian process, **61**, 78
  Markovian, 62
  zero mean, 61
generalized feedback shift register generator, 15
genetic algorithm, 149
geometric Brownian motion, **80–81**
geometric cooling in simulated annealing, 146
geometric distribution, **46–47**
Gibbs sampler, **91**
  grouped, 95
  random sweep, 93
  reversible, 93
  systematic, 93
gradient
  estimation, **123**
grouped Gibbs sampler, 95

Hammersley–Clifford theorem, 92
hierarchical model, 93
hit-and-run sampler, **95**
holding rate, 67
hyperball, 40
hypersphere, 41

iid
  sequence, 9
importance sampling, **113–155**
  density, 113
    optimal, **114**, 115, 156
  estimator, 113, 156
increment
  —s independent, 73
  of an LCG, 12
independence sampler, 88–89
independent increments, 73
indicator, 157
infinitesimal perturbation analysis, **128**, 137
information matrix, 90
instrumental density, *see* proposal density
intensity function, 69
inverse — distribution, **30**
  gamma, **30**
inverse-transform method, **26–28**, 39, 107
  for gradient estimation, 125
irreducible Markov chain, 93
Itô diffusion, *see* diffusion process
Itô integral, 76
iterative averaging, 139

joint distribution, 40

Kiefer–Wolfowitz algorithm, 139, 140
KISS99 generator, 17
Kolmogorov–Smirnov test, 19, 20
Kullback–Leibler distance, 117, 155

Langevin

Metropolis–Hastings sampler, 100
SDE, 83
likelihood
optimization, 157
ratio, 113, 132, 160
method, *see* score function method
limiting distribution, 87
line sampler, 95
linear congruential generator, 12–13
linear feedback shift register, 15
linear transformation, 28
location family, 29
location–scale family, **29–30**
logarithmic efficiency, 120
logit model, 89
Lorentz distribution, *see* Cauchy distribution

Markov chain, 87, 92
generation, 63–66
Markov chain Monte Carlo, **87–145**
Markov jump process, 71
generation, 66–68
Markov process, 62
Markov property, 63, 66, 73
MATLAB, 9
matrix congruential generator, 13
maximum likelihood estimator, 157, 165
maximum-of-d test, 22
mean measure, 69
Mersenne twister, 12, 15
Metropolis–Hastings algorithm, **87–91**, 93
mixture distribution, **32**
of normals, 33
modulo 2 linear generator, **14–15**
modulus
of a matrix congruential generator, 13
of an LCG, 12
of an MRG, 13
Monte Carlo
crude, 104, 128
MRG32k3a random number generator, 11, 16, 17
MT19937 Mersenne twister, 12, 15
multinormal distribution, *see* multivariate normal distribution
multiple control variable, 110
multiple-recursive generator, 13
combined, 12, 16, 17
multiplicative congruential generator, 12, 16
matrix, 13
minimal standard, 12
multiplier
of an LCG, 12
of an MRG, 13
multivariate Gaussian distribution, *see* multivariate normal distribution
multivariate normal distribution, **56–58**, 58, 61
multivariate Student's t distribution, **58**

negative exponential distribution, *see* exponential distribution
Newton's method, 90
noisy optimization, 137, 168–169
nominal distribution, 113
nominal parameter, 115
normal antithetic random variables, 107
normal distribution, 29, 39, **53–54**, 56–58
mixture, 33
positive, **39**, 54
normal updating
in cross-entropy, 163

objective function, 137
optimal importance sampling density, **114**
optimization
combinatorial, 161
constrained, 165
continuous, 163
noisy, 137, 168–169

randomized, **137–159**
order of an MRG, 13
Ornstein–Uhlenbeck process, **82–83**

p-value, 24
partition test, 21
penalty function, 165
performance
measure, 123
period
length, of a random number generator, 10
permutation test, 22
Poisson
zero inflated — model, 93
Poisson distribution, **47–48**
Poisson process, 47, **69**
non-homogeneous, 71
Poisson random measure, 69
poker test, 21
polar method, **34**, 53
Polyak
averaging, 139
positive definite matrix, 56
positive normal distribution, **39**, 54
positive semidefinite matrix, 56
positivity condition, 92
prior pdf
improper, 100
process
Gaussian, **61**, 78
Markovian, 62
zero mean, 61
Markov, 62
Markov chain, 87, 92
generation, 63–66
Markov jump, 71
generation, 66–68
Ornstein–Uhlenbeck, **82–83**
Poisson, 47, **69**
Wiener, **73–75**
multidimensional, 74
projected subgradient method, 138
proposal density, 38, 87, 147
pseudorandom number, *see* random number
push-out method, 125

Q-matrix, 66

random
counting measure, 69
directions algorithm, 139
number, 10
generation, 9
tests for —s, 17–24
permutation, 41
process
generation, 61
sweep Gibbs sampler, 93
vector
generation, 39
walk, **65**, 73
on an n-cube, 65
sampler, **89–91**, 146
randomized optimization, **137–159**
Rao–Blackwellization, 111
rare-event
probability, 118
simulation
via cross-entropy, 157
rate function, 69
reciprocal distribution, **30**
reference parameter, **115**, 117, 156, 157
reliability, 67–68
repairman problem, 67–68
resampling, 35
without replacement, 42
response surface estimation, 133–135
reversible
Gibbs sampler, 93
Markov chain, 93
Robbins–Monro algorithm, 139, 140
root finding, 123
Rosenbrock function, 151
run test, 22
runs above/below the mean, 21

sample
average approximation, *see* stochastic counterpart method

performance function, 156
sampling, *see* resampling
satisfiability problem (SAT), 150, 161–163
scale family, **29–52**
scaling factor in differential evolution, 151
score function, 130
estimator, 130, 132
method, **129–135**, 137
r-th order, 131
seed of a random number generator, 9
sensitivity analysis, 123–129
serial test, 21
simulated annealing, 87, **145–148**
temperature, 145
smoothing parameter, 161
sparse matrix, 13
standard Brownian motion, 74
standard normal distribution, **53–54**
statistic, 18
statistical test
for random number generators, 17–24
Stirling number of the second kind, 21
stochastic approximation, **137–142**
stochastic counterpart method, 115, 118, **142–145**, 156
stochastic differential equation, **75–78**
for geometric Brownian motion, 80
for Ornstein–Uhlenbeck process, 82
generation via Euler, 76
multidimensional, 76
stochastic exponential, 80
stochastic optimization, *see* noisy optimization
stochastic process
generation, 61
stochastic shortest path, 104, 106, 109, 113, 118, 127
structural parameter, **123**, 125, 128
Student's t distribution, 58, 90
subgradient method, 138
systematic Gibbs sampler, 93

table lookup method, 34–35, 48
Tausworthe generator, 15
temperature in simulated annealing, 145
test
binary rank, 18
birthday spacings, 23
χ^2, 20
collision, 23
coupon collector's, 22
equidistribution, **20**
frequency, **20**
gap, **21**
Kolmogorov–Smirnov, 20
maximum-of-d, 22
partition, 21
permutation, 22
poker, 21
rank, binary, 23
run, 22
serial, 21
TestU01, 20
theorem
Hammersley–Clifford, 92
tilting vector, **115**
transformation
methods for generation, 28
transition
matrix, 63, 67
rate, 66
trigonometric function, 146
truncated
distribution, **30**
exponential distribution, 31
multivariate normal distribution, 98
normal distribution, 31

uniform distribution, 27, **54**
discrete, **48**
for permutations, 41
in a hypersphere, 40
uniform random number

generation, 9

variance
reduction, 103
variance minimization method, **115–117**

weak derivative, 137
white noise
Gaussian, 80
whitening, 57
Wichman–Hill generator, 16
Wiener process, **73–75**
multidimensional, 74

XOR, 14

zero inflated Poisson model, 93
zero-variance importance sampling distribution, 118