2

Random Variable Generation

> "Have you any thought," resumed Valentin, "of a tool with which it could be done?"
> "Speaking within modern probabilities, I really haven't," said the doctor.
> —G.K. Chesterton, *The Innocence of Father Brown*

The methods developed in this book mostly rely on the possibility of producing (with a computer) a supposedly endless flow of random variables (usually iid) for well-known distributions. Such a simulation is, in turn, based on the production of uniform random variables. Although we are not directly concerned with the *mechanics* of producing uniform random variables (see Note 2.6.1), we are concerned with the *statistics* of producing uniform and other random variables.

In this chapter we first consider what statistical properties we want a sequence of simulated uniform random variables to have. Then we look at some basic methodology that can, starting from these simulated uniform random variables, produce random variables from both standard and nonstandard distributions.

2.1 Introduction

Methods of simulation are based on the production of random variables, originally independent random variables, that are distributed according to a distribution f that is not necessarily explicitly known (see, for example, Examples 1.1, 1.2, and 1.3). The type of random variable production is formalized below in the definition of a *pseudo-random number generator*. We first concentrate on the generation of random variables that are uniform on the interval $[0, 1]$, because the uniform distribution $\mathcal{U}_{[0,1]}$ provides the basic probabilistic representation of randomness and also because all other distributions require a sequence of uniform variables to be simulated.

C. P. Robert et al., *Monte Carlo Statistical Methods*

2.1.1 Uniform Simulation

The logical paradox[1] associated with the generation of "random numbers" is the problem of producing a *deterministic* sequence of values in $[0,1]$ which imitates a sequence of *iid* uniform random variables $\mathcal{U}_{[0,1]}$. (Techniques based on the physical imitation of a "random draw" using, for example, the internal clock of the machine have been ruled out. This is because, first, there is no guarantee on the *uniform* nature of numbers thus produced and, second, there is no reproducibility of such samples.) However, we really do not want to enter here into the philosophical debate on the notion of "random," and whether it is, indeed, possible to "reproduce randomness" (see, for example, Chaitin 1982, 1988).

For our purposes, there are methods that use a fully deterministic process to produce a random sequence in the following sense: Having generated $(X_1, \ldots, X_n)$, knowledge of X_n [or of $(X_1, \ldots, X_n)$] imparts no discernible knowledge of the value of X_{n+1} if the transformation function is not available. Of course, given the initial value X_0 and the transformation function, the sample $(X_1, \ldots, X_n)$ is always the same. Thus, the "pseudo-randomness" produced by these techniques is limited since two samples $(X_1, \ldots, X_n)$ and $(Y_1, \ldots, Y_n)$ produced by the algorithm will not be independent, nor identically distributed, nor comparable in any probabilistic sense. This limitation should not be forgotten: The validity of a random number generator is based on a single sample $X_1, \ldots, X_n$ when n tends to $+\infty$ and not on replications $(X_{11}, \ldots, X_{1n}), (X_{21}, \ldots, X_{2n}), \ldots (X_{k1}, \ldots, X_{kn})$, where n is fixed and k tends to infinity. In fact, the distribution of these n-tuples depends only on the manner in which the initial values X_{r1} $(1 \leq r \leq k)$ were generated.

With these limitations in mind, we can now introduce the following operational definition, which avoids the difficulties of the philosophical distinction between a deterministic algorithm and the reproduction of a random phenomenon.

Definition 2.1. *A uniform pseudo-random number generator* is an algorithm which, starting from an initial value u_0 and a transformation D, produces a sequence $(u_i) = (D^i(u_0))$ of values in $[0,1]$. For all n, the values $(u_1, \ldots, u_n)$ reproduce the behavior of an iid sample $(V_1, \ldots, V_n)$ of uniform random variables when compared through a usual set of tests.

This definition is clearly restricted to *testable* aspects of the random variable generation, which are connected through the deterministic transformation

[1] Von Neumann (1951) summarizes this problem very clearly by writing *"Any one who considers arithmetical methods of reproducing random digits is, of course, in a state of sin. As has been pointed out several times, there is no such thing as a random number—there are only methods of producing random numbers, and a strict arithmetic procedure of course is not such a method."*

$u_i = D(u_{i-1})$. Thus, the validity of the algorithm consists in the verification that the sequence $U_1, \ldots, U_n$ leads to acceptance of the hypothesis

$$\mathrm{H}_0 : U_1, \ldots, U_n \quad \text{are iid} \quad \mathcal{U}_{[0,1]}.$$

The set of tests used is generally of some consequence. There are classical tests of uniformity, such as the Kolmogorov–Smirnov test. Many generators will be deemed adequate under such examination. In addition, and perhaps more importantly, one can use methods of *time series* to determine the degree of correlation between between U_i and $(U_{i-1}, \ldots, U_{i-k})$, by using an ARMA$(p, q)$ model, for instance. One can use nonparametric tests, like those of Lehmann (1975) or Randles and Wolfe (1979), applying them on arbitrary decimals of U_i. Marsaglia[2] has assembled a set of tests called `Die Hard`.

Definition 2.1 is therefore *functional*: An algorithm that generates uniform numbers is acceptable if it is not rejected by a set of tests. This methodology is not without problems, however. Consider, for example, particular applications that might demand a large number of iterations, as the theory of large deviations (Bucklew 1990), or particle physics, where algorithms resistant to standard tests may exhibit fatal faults. In particular, algorithms having hidden periodicities (see below) or which are not uniform for the smaller digits may be difficult to detect. Ferrenberg et al. (1992) show, for instance, that an algorithm of Wolff (1989), reputed to be "good," results in systematic biases in the processing of Ising models (see Example 5.8), due to long-term correlations in the generated sequence.

The notion that a deterministic system can imitate a random phenomenon may also suggest the use of *chaotic* models to create random number generators. These models, which result in complex deterministic structures (see Bergé et al. 1984, Gleick 1987, Ruelle 1987) are based on dynamic systems of the form $X_{n+1} = D(X_n)$ which are very sensitive to the initial condition X_0.

Example 2.2. The logistic function. The logistic function $D_\alpha(x) = \alpha x(1-x)$ produces, for some values of $\alpha \in [3.57, 4.00]$, chaotic configurations. In particular, the value $\alpha = 4.00$ yields a sequence (X_n) in $[0, 1]$ that, theoretically, has the same behavior as a sequence of random numbers (or random variables) distributed according to the *arcsine distribution* with density $1/\pi\sqrt{x(1-x)}$. (See Problem 2.4 for another random number generator based on the "tent" function.)

Although the limit distribution (also called the stationary distribution) associated with a dynamic system $X_{n+1} = D(X_n)$ is sometimes defined and known, the chaotic features of the system do not guarantee acceptable behavior (in the probabilistic sense) of the associated generator. Figure 2.1 illustrates the properties of the generator based on the logistic function D_α. The

[2] These tests are now available as a freeware on the site `http://stat.fsu.edu/~geo/diehard.html`

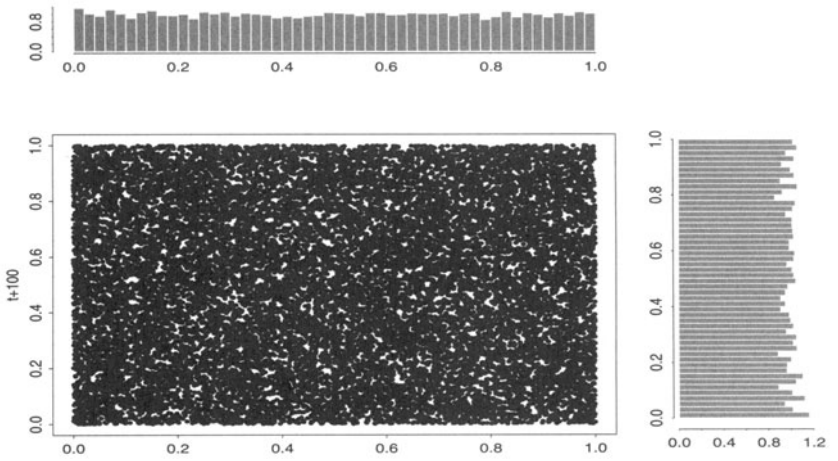

Fig. 2.1. Plot of the sample (y_n, y_{n+100}) $(n = 1, \ldots, 9899)$ for the sequence $x_{n+1} = 4x_n(1 - x_n)$ and $y_n = F(x_n)$, along with the (marginal) histograms of y_n *(on top)* and y_{n+100} *(right margin)*.

histogram of the transformed variables $Y_n = 0.5 + \arcsin(X_n)/\pi$, of a sample of successive values $X_{n+1} = D_\alpha(X_n)$ fits the uniform density extremely well. Moreover, while the plots of (Y_n, Y_{n+1}) and (Y_n, Y_{n+10}) do not display characteristics of uniformity, Figure 2.1 shows that the sample of (Y_n, Y_{n+100}) satisfactorily fills the unit square. However, even when these functions give a good approximation of randomness in the unit square $[0,1] \times [0,1]$, the hypothesis of randomness is rejected by many standard tests.

Classic examples from the theory of chaotic functions do not lead to acceptable pseudo-random number generators. Moreover, the 100 calls to D_α between two generations are excessive in terms of computing time. ‖

We have presented in this introduction some necessary basic notions to now understand a very good pseudo-random number generator, the algorithm Kiss[3] of Marsaglia and Zaman (1993). However, many of the details involve notions that are a bit tangential to the main topic of this text and, in addition, most computer packages now include a well-behaved uniform random generator. Thus, we leave the details of the Kiss generator to Note 2.6.1.

2.1.2 The Inverse Transform

In describing the structure of a space of random variables, it is always possible to represent the generic probability triple $(\Omega, \mathcal{F}, P)$ (where Ω represents the whole space, $\mathcal{F}$ represents a σ-algebra on Ω, and P is a probability measure) as

[3] The name is an acronym of the saying *Keep it simple, stupid!*, and not reflective of more romantic notions. After all, this is a Statistics text!

$([0,1], \mathcal{B}, \mathcal{U}_{[0,1]})$ (where $\mathcal{B}$ are the Borel sets on $[0,1]$) and therefore equate the variability of $\omega \in \Omega$ with that of a uniform variable in $[0,1]$ (see, for instance, Billingsley 1995, Section 2). The random variables X are then functions from $[0,1]$ to $\mathcal{X}$, that is, functions of uniform variates transformed by the *generalized inverse* function.

Definition 2.3. For a non-decreasing function F on $\mathbb{R}$, the *generalized inverse* of F, F^-, is the function defined by

$$F^-(u) = \inf\{x : \ F(x) \geq u\} \ . \tag{2.1}$$

We then have the following lemma, sometimes known as the *probability integral transform*, which gives us a representation of any random variable as a transform of a uniform random variable.

Lemma 2.4. *If $U \sim \mathcal{U}_{[0,1]}$, then the random variable $F^-(U)$ has the distribution F.*

Proof. For all $u \in [0,1]$ and for all $x \in F^-([0,1])$, the generalized inverse satisfies

$$F(F^-(u)) \geq u \quad \text{and} \quad F^-(F(x)) \leq x \ .$$

Therefore,

$$\{(u,x) : \ F^-(u) \leq x\} = \{(u,x) : \ F(x) \geq u\}$$

and

$$P(F^-(U) \leq x) = P(U \leq F(x)) = F(x) \ .$$

□

Thus, formally, in order to generate a random variable $X \sim F$, it suffices to generate U according to $\mathcal{U}_{[0,1]}$ and then make the transformation $x = F^-(u)$.

Example 2.5. Exponential variable generation. If $X \sim \mathcal{E}xp(1)$, so $F(x) = 1 - e^{-x}$, then solving for x in $u = 1 - e^{-x}$ gives $x = -\log(1-u)$. Therefore, if $U \sim \mathcal{U}_{[0,1]}$, the random variable $X = -\log U$ has the exponential distribution (as U and $1 - U$ are both uniform). ‖

The generation of uniform random variables is therefore a key determinant in the behavior of simulation methods for other probability distributions, since those distributions can be represented as a deterministic transformation of uniform random variables. (Although, in practice, we often use methods other than that of Lemma 2.4, this basic representation is usually a good way to think about things. Note also that Lemma 2.4 implies that a bad choice of a uniform random number generator can invalidate the resulting simulation procedure.)

As mentioned above, from a theoretical point of view an operational version of any probability space $(\Omega, \mathcal{A}, P)$ can be created from the uniform distribution $\mathcal{U}_{[0,1]}$ and Lemma 2.4. Thus, the generation of any sequence of random variables can be formally implemented through the uniform generator Kiss. In practice, however, this approach only applies when the cumulative distribution functions are "explicitly" available, in the sense that there exists an algorithm allowing the computation of $F^-(u)$ in acceptable time. In particular, for distributions with explicit forms of F^- (for instance, the exponential, double-exponential, or Weibull distributions; see Problem 2.5 for other examples), Lemma 2.4 does lead to a practical implementation. But this situation only covers a small number of cases, described in Section 2.2 and additional problems. Other methods, like the Accept–Reject method of Section 2.3, are more general and do not use any strong analytic property of the densities. Thus, they can handle more general cases as, for example, the simulation of distributions in dimensions greater than one.

2.1.3 Alternatives

Although computation by Monte Carlo methods can be thought of as an exact calculation (as the order of accuracy is only a function of computation time), it is probably more often thought of as an approximation. Thus, numerical approximation is an alternative to Monte Carlo, and should also be considered a candidate for solving any particular problem. The following example shows how numerical approximations can work in the calculation of normal probabilities (see Sections 1.4, 3.6.2 and 3.4 for other approaches).

Example 2.6. Normal probabilities. Although Φ, the cumulative distribution function of the normal distribution cannot be expressed explicitly, since

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\{-z^2/2\} dz,$$

there exist approximations of Φ and of Φ^{-1} up to an arbitrary precision. For instance, Abramowitz and Stegun (1964) give the approximation

$$\Phi(x) \simeq 1 - \varphi(x) \left[b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5\right] \qquad (x > 0)\,,$$

where φ denotes the normal density, $t = (1 + px)^{-1}$ and

$$\begin{aligned} p &= 0.2316419, \quad & b_1 &= 0.31938, \quad & b_2 &= -0.35656, \\ b_3 &= 1.78148, \quad & b_4 &= -1.82125, \quad & b_5 &= 1.33027. \end{aligned}$$

Similarly, we also have the approximation

$$\Phi^{-1}(\alpha) \simeq t - \frac{a_0 + a_1 t}{1 + b_1 t + b_2 t^2},$$

where $t^2 = \log(\alpha^{-2})$ and

$$a_0 = 2.30753, \quad a_1 = 0.27061, \quad b_1 = 0.99229, \quad b_2 = 0.04481.$$

These two approximations are exact up to an error of order 10^{-8}, the error being absolute. If no other fast simulation method was available, this approximation could be used in settings which do not require much precision in the tails of $\mathcal{N}(0,1)$. (However, as shown in Example 2.8, there exists an exact and much faster algorithm.) ||

2.1.4 Optimal Algorithms

Devroye 1985 presents a more comprehensive (one could say almost exhaustive!) treatment of the methods of random variable generation than the one presented in this chapter, in particular looking at refinements of existing algorithms in order to achieve uniformly optimal performances. (We strongly urge the reader to consult this book[4] for a better insight on the implications of this goal in terms of probabilistic and algorithmic complexity.)

Some refinements of the simulation techniques introduced in this chapter will be explored in Chapter 4, where we consider ways to accelerate Monte Carlo methods. At this point, we note that the concepts of "optimal" and "efficient" algorithms are particularly difficult to formalize. We can naively compare two algorithms, $[B_1]$ and $[B_2]$ say, in terms of time of computation, for instance through the average generation time of one observation. However, such a comparison depends on many subjective factors like the quality of the programming, the particular programming language used to implement the method, and the particular machine on which the program runs. More importantly, it does not take into account the conception and programming (and debugging) times, nor does it incorporate the specific use of the sample produced, partly because a quantification of these factors is generally impossible. For instance, some algorithms have a decreasing efficiency when the sample size increases. The reduction of the efficiency of a given algorithm to its average computation time is therefore misleading and we only use this type of measurement in settings where $[B_1]$ and $[B_2]$ are already of the same complexity. Devroye (1985) also notes that the simplicity of algorithms should be accounted for in their evaluation, since complex algorithms facilitate programming errors and, therefore, may lead to important time losses.[5]

A last remark to bring this section to its end is that simulation of the standard distributions presented here is accomplished quite efficiently by many statistical programming packages (for instance, `Gauss`, `Mathematica`, `Matlab`, `R`, `Splus`). When the generators from these general-purpose packages are easily accessible (in terms of programming), it is probably preferable to use such

[4] The book is now out-of-print but available for free on the author's website, at McGill University, Montréal, Canada.

[5] In fact, in numerous settings, the time required by a simulation is overwhelmingly dedicated to programming. This is, at least, the case for the authors themselves!!!

a generator rather than to write one's own. However, if a generation technique will get extensive use or if there are particular features of a problem that can be exploited, the creation of a personal library of random variable generators can accelerate analyses and even improve results, especially if the setting involves "extreme" cases (sample size, parameter values, correlation structure, rare events) for which the usual generators are poorly adapted. The investment represented by the creation and validation of such a personal library must therefore be weighed against the potential benefits.

2.2 General Transformation Methods

When a distribution f is linked in a relatively simple way to another distribution that is easy to simulate, this relationship can often be exploited to construct an algorithm to simulate variables from f. In this section we present alternative (to Lemma 2.4) techniques for generating nonuniform random variables. Some of these methods are rather case-specific, and are difficult to generalize as they rely on properties of the distribution under consideration and its relation with other probability distributions.

We begin with an illustration of some distributions that are simple to generate.

Example 2.7. Building on exponential random variables. In Example 2.5 we saw how to generate an exponential random variable starting from a uniform. Now we illustrate some of the random variables that can be generated starting from an exponential distribution. If the X_i's are iid $\mathcal{E}xp(1)$ random variables, then

$$
\begin{aligned}
&Y = 2\sum_{j=1}^{\nu} X_j \sim \chi^2_{2\nu}\,, \qquad \nu \in \mathbb{N}^*\,,\\
&Y = \beta\sum_{j=1}^{a} X_j \sim \mathcal{G}a(a,\beta)\,, \qquad a \in \mathbb{N}^*\,,\\
&Y = \frac{\sum_{j=1}^{a} X_j}{\sum_{j=1}^{a+b} X_j} \sim \mathcal{B}e(a,b)\,, \qquad a,b \in \mathbb{N}^*\,.
\end{aligned}
\tag{2.2}
$$

Other derivations are possible (see Problem 2.6). ‖

These transformations are quite simple to use and, hence, will often be a favorite. However, there are limits to their usefulness, both in scope of variables that can be generated and in efficiency of generation. For example, as we will see, there are more efficient algorithms for Gamma and Beta random variables. Also, we cannot use exponentials to generate Gamma random variables with a non-integer shape parameter. For instance, we cannot get a χ^2_1

variable, which would, in turn, get us a $\mathcal{N}(0,1)$ variable. For that, we look at the following example of the *Box–Muller* algorithm (1958) for the generation of $\mathcal{N}(0,1)$ variables.

Example 2.8. Normal variable generation. If r and θ are the polar coordinates of (X_1, X_2), then, since the distribution of (X_1, X_2) is rotation invariant (see Problem 2.7)

$$r^2 = X_1^2 + X_2^2 \sim \chi_2^2 = \mathcal{E}xp(1/2)\,,$$
$$\theta \sim \mathcal{U}_{[0,2\pi]}\,.$$

If U_1 and U_2 are iid $\mathcal{U}_{[0,1]}$, the variables X_1 and X_2 defined by

$$X_1 = \sqrt{-2\log(U_1)}\,\cos(2\pi U_2)\,, \qquad X_2 = \sqrt{-2\log(U_1)}\,\sin(2\pi U_2)\,,$$

are then iid $\mathcal{N}(0,1)$. The corresponding algorithm is

Algorithm A.3 –Box–Muller–

1 Generate U_1, U_2 iid $\mathcal{U}_{[0,1]}$;
2 Define [A.3]

$$\begin{cases} x_1 = \sqrt{-2\log(u_1)}\cos(2\pi u_2)\,, \\ x_2 = \sqrt{-2\log(u_1)}\sin(2\pi u_2)\,; \end{cases}$$

3 Take x_1 and x_2 as two independent draws from $\mathcal{N}(0,1)$.

In comparison with algorithms based on the Central Limit Theorem, this algorithm is exact, producing two normal random variables from two uniform random variables, the only drawback (in speed) being the necessity of calculating functions such as log, cos, and sin. If this is a concern, Devroye (1985) gives faster alternatives that avoid the use of these functions (see also Problems 2.8 and 2.9). ||

Example 2.9. Poisson generation. The Poisson distribution is connected to the exponential distribution through the Poisson process; that is, if $N \sim \mathcal{P}(\lambda)$ and $X_i \sim \mathcal{E}xp(\lambda)$, $i \in \mathbb{N}^*$, then

$$P_\lambda(N = k) = P_\lambda(X_1 + \cdots + X_k \leq 1 < X_1 + \cdots + X_{k+1})\,.$$

Thus, the Poisson distribution can be simulated by generating exponential random variables until their sum exceeds 1. This method is simple, but is really practical only for smaller values of λ. On average, the number of exponential variables required is λ, and this could be prohibitive for large values of λ. In these settings, Devroye (1981) proposed a method whose computation time

is uniformly bounded (in λ) and we will see another approach, suitable for large λ's, in Example 2.23. Note also that a generator of Poisson random variables can produce negative binomial random variables since, when $Y \sim \mathcal{G}a(n, (1-p)/p)$ and $X|y \sim \mathcal{P}(y)$, $X \sim \mathcal{N}eg(n,p)$. (See Problem 2.13.) ||

Example 2.9 shows a specific algorithm for the generation of Poisson random variables. Based on an application of Lemma 2.4, we can also construct a generic algorithm that will work for any discrete distribution.

Example 2.10. Discrete random variables. To generate $X \sim P_\theta$, we can calculate (once for all) the probabilities

$$p_0 = P_\theta(X \leq 0), \quad p_1 = P_\theta(X \leq 1), \quad p_2 = P_\theta(X \leq 2), \quad \ldots$$

then generate $U \sim \mathcal{U}_{[0,1]}$ and take

$$X = k \text{ if } p_{k-1} < U < p_k.$$

For example, to generate $X \sim \mathcal{B}in(10, .3)$, the first values are

$$p_0 = 0.028, \quad p_1 = 0.149, \quad p_2 = 0.382, \ldots, p_{10} = 1\,,$$

and to generate $X \sim \mathcal{P}(7)$, take

$$p_0 = 0.0009, \quad p_1 = 0.0073, \quad p_2 = 0.0296, \ldots$$

the sequence being stopped when it reaches 1 with a given number of decimals. (For instance, $p_{20} = 0.999985$.) Specific algorithms, such as Example 2.9, are usually more efficient but it is mostly because of the storage problem. See Problem 2.12 and Devroye (1985). ||

Example 2.11. Beta generation. Consider $U_1, \ldots, U_n$, an iid sample from $\mathcal{U}_{[0,1]}$. If $U_{(1)} \leq \cdots \leq U_{(n)}$ denotes the ordered sample, that is, the *order statistics* of the original sample, $U_{(i)}$ is distributed as $\mathcal{B}e(i, n-i+1)$ and the vector of the differences $(U_{(i_1)}, U_{(i_2)} - U_{(i_1)}, \ldots, U_{(i_k)} - U_{(i_{k-1})}, 1 - U_{(i_k)})$ has a Dirichlet distribution $\mathcal{D}(i_1, i_2 - i_1, \ldots, n - i_k + 1)$ (see Problem 2.17). However, even though these probabilistic properties allow the direct generation of Beta and Dirichlet random variables from uniform random variables, they do not yield efficient algorithms. The calculation of the order statistics can, indeed, be quite time-consuming since it requires sorting the original sample. Moreover, it only applies for integer parameters in the Beta distribution.

The following result allows for an alternative generation of Beta random variables from uniform random variables: Jöhnk's Theorem (see Jöhnk 1964 or Devroye 1985) states that if U and V are iid $\mathcal{U}_{[0,1]}$, the distribution of

$$\frac{U^{1/\alpha}}{U^{1/\alpha} + V^{1/\beta}},$$

conditional on $U^{1/\alpha} + V^{1/\beta} \leq 1$, is the $\mathcal{B}e(\alpha, \beta)$ distribution. However, given the constraint on $U^{1/\alpha} + V^{1/\beta}$, this result does not provide a good algorithm to generate $\mathcal{B}e(\alpha, \beta)$ random variables for large values of α and β, as shown by the fast decrease of the probability of accepting a pair (U, V) as a function of $\alpha = \beta$ in Figure 2.2. ||

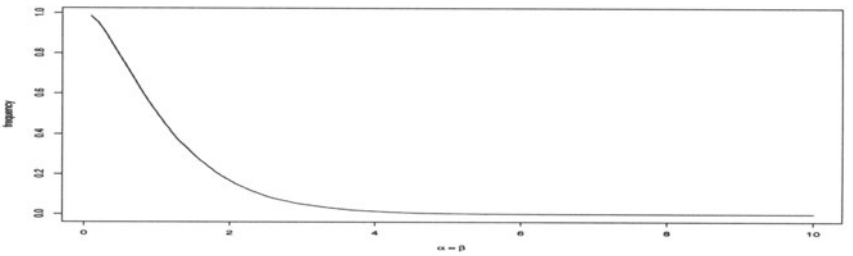

Fig. 2.2. Probability of accepting a pair (U, V) in Jöhnk (1964) algorithm as a function of α, when $\alpha = \beta$.

Example 2.12. Gamma generation. Given a generator of Beta random variables, we can derive a generator of Gamma random variables $\mathcal{G}a(\alpha, 1)$ $(\alpha < 1)$ the following way: If $Y \sim \mathcal{B}e(\alpha, 1-\alpha)$ and $Z \sim \mathcal{E}xp(1)$, then $X = YZ \sim \mathcal{G}a(\alpha, 1)$. Indeed, by making the transformation $x = yz, w = z$ and integrating the joint density, we find

$$\begin{aligned} f(x) &= \frac{\Gamma(1)}{\Gamma(\alpha)\Gamma(1-\alpha)} \int_x^\infty \left(\frac{x}{w}\right)^{\alpha-1} \left(1 - \frac{x}{w}\right)^{-\alpha} w^{-1} e^{-w} dw \\ &= \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}. \end{aligned} \tag{2.3}$$

Alternatively, if we can start with a Gamma random variable, a more efficient generator for $\mathcal{G}a(\alpha, 1)$ $(\alpha < 1)$ can be constructed: If $Y \sim \mathcal{G}a(\alpha + 1, 1)$ and $U \sim \mathcal{U}_{[0,1]}$, independent, then $X = YU^{1/\alpha}$ is distributed according to $\mathcal{G}a(\alpha, 1)$, since

$$f(x) \propto \int_x^\infty w^\alpha e^{-w} \left(\frac{x}{w}\right)^{\alpha-1} w^{-1} dw = x^{\alpha-1} e^{-x}. \tag{2.4}$$

(See Stuart 1962 or Problem 2.14). ||

The representation of a probability density as in (2.3) is a particular case of a *mixture of distributions*. Not only does such a representation induce relatively efficient simulation methods, but it is also related to methods in Chapters 9 and 10. The principle of a mixture representation is to write a density f as the marginal of another distribution, in the form

$$f(x) = \int_{\mathcal{Y}} g(x,y)\,dy \qquad \text{or} \qquad f(x) = \sum_{i \in \mathcal{Y}} p_i\, f_i(x)\,, \tag{2.5}$$

depending on whether $\mathcal{Y}$ is continuous or discrete. For instance, if the joint distribution $g(x,y)$ is simple to simulate, then the variable X can be obtained as a component of the generated (X,Y). Alternatively, if the component distributions $f_i(x)$ can be easily generated, X can be obtained by first choosing f_i with probability p_i and then generating an observation from f_i.

Example 2.13. Student's t generation. A useful form of (2.5) is

$$f(x) = \int_{\mathcal{Y}} g(x,y)\,dy = \int_{\mathcal{Y}} h_1(x|y)h_2(y)\,dy\,, \tag{2.6}$$

where h_1 and h_2 are the conditional and marginal densities of $X|Y=y$ and Y, respectively. For example, we can write Student's t density with ν degrees of freedom in this form, where

$$X|y \sim \mathcal{N}(0, \nu/y) \quad \text{and} \quad Y \sim \chi^2_\nu.$$

Such a representation is also useful for discrete distributions. In Example 2.9, we noted an alternate representation for the negative binomial distribution. If X is negative binomial, $X \sim \mathcal{N}eg(n,p)$, then $P(X=x)$ can be written as (2.6) with

$$X|y \sim \mathcal{P}(y) \quad \text{and} \quad Y \sim \mathcal{G}(n,\beta),$$

where $\beta = (1-p)/p$. Note that the discreteness of the negative binomial distribution does not result in a discrete mixture representation of the probability. The mixture is continuous, as the distribution of Y is itself continuous. ‖

Example 2.14. Noncentral chi squared generation. The noncentral chi squared distribution, $\chi^2_p(\lambda)$, also allows for a mixture representation, since it can be written as a sum of central chi squared densities. In fact, it is of the form (2.6) with h_1 the density of a χ^2_{p+2K} distribution and h_2 the density of $\mathcal{P}(\lambda/2)$. However, this representation is not as efficient as the algorithm obtained by generating $Z \sim \chi^2_{p-1}$ and $Y \sim \mathcal{N}(\sqrt{\lambda}, 1)$, and using the fact that $Z + Y^2 \sim \chi^2_p(\lambda)$. Note that the noncentral chi squared distribution does not have an explicit form for its density function. It is either represented as an infinite mixture (see (3.31)) or by using modified Bessel functions (see Problem 1.8). ‖

In addition to the above two examples, other distributions can be represented as mixtures (see, for instance, Gleser 1989). In many cases this representation can be exploited to produce algorithms for random variable generation (see Problems 2.24–2.26, and Note 2.6.3).

2.3 Accept–Reject Methods

There are many distributions from which it is difficult, or even impossible, to directly simulate by an inverse transform. Moreover, in some cases, we are not even able to represent the distribution in a usable form, such as a transformation or a mixture. In such settings, it is impossible to exploit direct probabilistic properties to derive a simulation method. We thus turn to another class of methods that only requires us to know the functional form of the density f of interest up to a multiplicative constant; no deep analytical study of f is necessary. The key to this method is to use a simpler (simulationwise) density g from which the simulation is actually done. For a given density g—called the *instrumental density*— there are thus many densities f—called the *target densities*—which can be simulated this way. The corresponding algorithm, called *Accept–Reject*, is based on a simple connection with the uniform distribution, discussed below.

2.3.1 The Fundamental Theorem of Simulation

There exists a fundamental (simple!) idea that underlies the Accept–Reject methodology, and also plays a key role in the construction of the slice sampler (Chapter 8). If f is the density of interest, on an arbitrary space, we can write

$$f(x) = \int_0^{f(x)} du\,. \tag{2.7}$$

Thus, f appears as the *marginal density* (in X) of the joint distribution,

$$(X,U) \sim \mathcal{U}\{(x,u) : 0 < u < f(x)\}\,. \tag{2.8}$$

Since U is not directly related to the original problem, it is called an *auxiliary variable*, a notion to be found again in later chapters like Chapters 8–10.

Although it seems like we have not gained much, the introduction of the auxiliary uniform variable in (2.7) has brought a considerably different perspective: Since (2.8) is the joint density of X and U, we can generate from this joint distribution by just generating uniform random variables on the constrained set $\{(x,u) : 0 < u < f(x)\}$. Moreover, since the marginal distribution of X *is* the original target distribution, f, by generating a uniform variable on $\{(x,u) : 0 < u < f(x)\}$, we have generated a random variable from f. And this generation was produced without using f other than through the calculation of $f(x)$! The importance of this equivalence is stressed in the following theorem:

Theorem 2.15 (Fundamental Theorem of Simulation). *Simulating*

$$X \sim f(x)$$

is equivalent to simulating

$$(X,U) \sim \mathcal{U}\{(x,u) : 0 < u < f(x)\}\,.$$

While this theorem is fundamental in many respects, it appears mostly as a formal representation at this stage because the simulation of the uniform pair (X, U) is often not straightforward. For example, we could simulate $X \sim f(x)$ and $U|X = x \sim \mathcal{U}(0, f(x))$, but then this makes the whole representation useless. And the symmetric approach, which is to simulate U from its marginal distribution, and then X from the distribution conditional on $U = u$, does not often result in a feasible calculation. The solution is to simulate the entire pair (X, U) at once in a bigger set, where simulation is easier, and then take the pair if the constraint is satisfied.

For example, in a one-dimensional setting, suppose that

$$\int_a^b f(x)dx = 1$$

and that f is bounded by m. We can then simulate the random pair $(Y, U) \sim \mathcal{U}(0 < u < m)$ by simulating $Y \sim \mathcal{U}(a, b)$ and $U|Y = y \sim \mathcal{U}(0, m)$, and take the pair only if the further constraint $0 < u < f(y)$ is satisfied. This results in the correct distribution of the accepted value of Y, call it X, because

$$\begin{aligned} P(X \le x) &= P(Y \le x | U < f(Y)) \\ &= \frac{\int_a^x \int_0^{f(y)} du\, dy}{\int_a^b \int_0^{f(y)} du\, dy} = \int_a^x f(y)\, dy. \end{aligned} \tag{2.9}$$

This amounts to saying that, if $A \subset B$ and if we generate a uniform sample on B, keeping only the terms of this sample that are in A will result in a uniform sample on A (with a random size that is independent of the values of the sample).

Example 2.16. Beta simulation. We have seen (Example 2.11) that direct simulation of Beta random variables can be difficult. However, we can easily use Theorem 2.15 for this simulation when $\alpha \ge 1$ and $\beta \ge 1$. Indeed, to generate $X \sim \mathcal{B}e(\alpha, \beta)$, we take $Y \sim \mathcal{U}_{[0,1]}$ and $U \sim \mathcal{U}_{[0,m]}$, where m is the maximum of the Beta density (Problem 2.15). For $\alpha = 2.7$ and $\beta = 6.3$ Figure 2.3 shows the results of generating 1000 pairs (Y, U). The pairs that fall under the density function are those for which we accept $X = Y$, and we reject those pairs that fall outside. ||

In addition, it is easy to see that the probability of acceptance of a given simulation in the box $[a, b] \times [0, m]$ is given by

$$P(\text{Accept}) = P(U < f(Y)) = \frac{1}{m} \int_0^1 \int_0^{f(y)} du\, dy = \frac{1}{m}.$$

For Example 2.16, $m = 2.67$, so we accept approximately $1/2.67 = 37\%$ of the values.

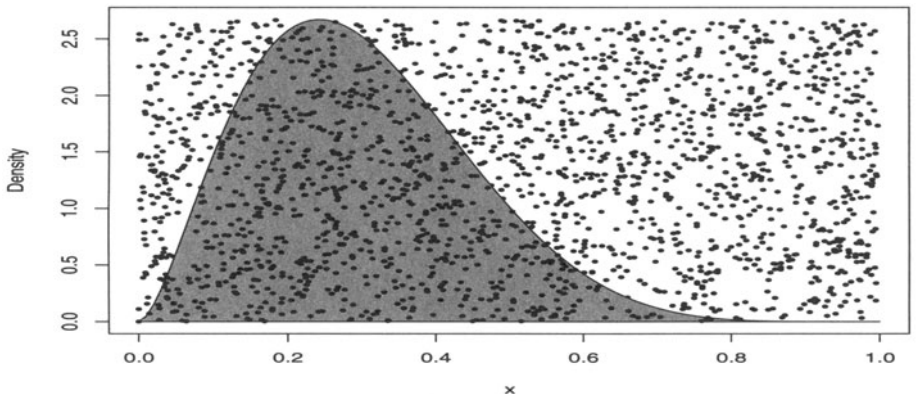

Fig. 2.3. Generation of Beta random variables: Using Theorem 2.15, 1000 (Y,U) pairs were generated, and 365 were accepted (the black circles under the Beta $\mathcal{B}e(2.7, 6.3)$ density function).

The argument leading to (2.9) can easily be generalized to the situation where the larger set is not a box any longer, as long as simulating uniformly over this larger set is feasible. This generalization may then allow for cases where either or both of the support of f and the maximum of f are unbounded. If the larger set is of the form

$$\mathscr{L} = \{(y,u) : 0 < u < m(y)\} ,$$

the constraints are thus that $m(x) \geq f(x)$ and that simulation of a uniform on $\mathscr{L}$ is feasible. Obviously, efficiency dictates that m be as close as possible to f in order to avoid wasting simulations. A remark of importance is that, because of the constraint $m(x) \geq f(x)$, m cannot be a probability density. We then write

$$m(x) = Mg(x) \text{ where } \int_{\mathcal{X}} m(x)\,dx = \int_{\mathcal{X}} Mg(x)\,dx = M ,$$

since m is necessarily integrable (otherwise, $\mathscr{L}$ would not have finite mass and a uniform distribution would not exist on $\mathscr{L}$). As mentioned above, a natural way of simulating the uniform on $\mathscr{L}$ is then to use (2.7) backwards, that is, to simulate $Y \sim g$ and then $U|Y = y \sim \mathcal{U}(0, Mg(y))$. If we only accept the y's such that the constraint $u < f(y)$ is satisfied, we have

$$\begin{aligned} P(X \in \mathcal{A}) &= P(Y \in \mathcal{A} | U < f(Y)) \\ &= \frac{\int_{\mathcal{A}} \int_0^{f(y)} \frac{du}{Mg(y)}\, g(y)dy}{\int_{\mathcal{A}} \int_0^{f(y)} \frac{du}{Mg(y)}\, g(y)dy} = \int_{\mathcal{A}} f(y)\, dy \end{aligned}$$

for every measurable set $\mathcal{A}$ and the accepted X's are indeed distributed from f. We have thus derived a more general implementation of the fundamental theorem, as follows:

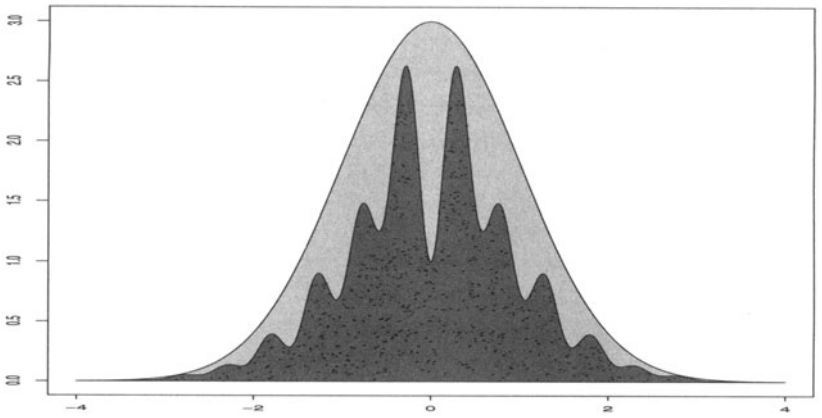

Fig. 2.4. Plot of a uniform sample over the set $\{(x,u) : 0 < u < f(x)\}$ for $f(x) \propto \exp(-x^2/2)(\sin(6x)^2 + 3\cos(x)^2\sin(4x)^2 + 1)$ and of the envelope function $g(x) = 5\exp(-x^2/2)$.

Corollary 2.17. *Let $X \sim f(x)$ and let $g(x)$ be a density function that satisfies $f(x) \leq Mg(x)$ for some constant $M \geq 1$. Then, to simulate $X \sim f$, it is sufficient to generate*

$$Y \sim g \qquad \text{and} \qquad U|Y = y \sim \mathcal{U}(0, Mg(y))\,,$$

until $0 < u < f(y)$.

Figure 2.4 illustrates Corollary 2.17 for the target density

$$f(x) \propto \exp(-x^2/2)(\sin(6x)^2 + 3\cos(x)^2\sin(4x)^2 + 1)$$

with upper bound (or, rather, dominating density) the normal density

$$g(x) = \exp(-x^2/2)/\sqrt{2\pi}\,,$$

which is obviously straightforward to generate.

Corollary 2.17 has two consequences. First, it provides a generic method to simulate from any density f that is known *up to a multiplicative factor*; that is, the normalizing constant of f need not be known, since the method only requires input of the ratio f/M, which does not depend on the normalizing constant. This is for instance the case of Figure 2.4, where the normalizing constant of f is unknown. This property is particularly important in Bayesian calculations. There, a quantity of interest is the posterior distribution, defined according to Bayes Theorem by

$$\pi(\theta|x) \propto \pi(\theta)\, f(x|\theta)\,. \tag{2.10}$$

Thus, the posterior density $\pi(\theta|x)$ is easily specified up to a normalizing constant and, to use Corollary 2.17, this constant need not be calculated. (See Problem 2.29.)

Of course, there remains the task of finding a density g satisfying $f \leq Mg$, a bound that need not be tight, in the sense that Corollary 2.17 remains valid when M is replaced with any larger constant. (See Problem 2.30.)

A second consequence of Corollary 2.17 is that the probability of acceptance is exactly $1/M$ (a geometric waiting time), when evaluated for the properly normalized densities, and the expected number of trials until a variable is accepted is M (see Problem 2.30). Thus, a comparison between different simulations based on different instrumental densities $g_1, g_2, \ldots$ can be undertaken through the comparison of the respective bounds $M_1, M_2, \ldots$ (as long as the corresponding densities $g_1, g_2, \ldots$ are correctly normalized). In particular, a first method of optimizing the choice of g in $g_1, g_2, \ldots$ is to find the smallest bound M_i. However, this first and rudimentary comparison technique has some limitations, which we will see later in this chapter.

2.3.2 The Accept–Reject Algorithm

The implementation of Corollary 2.17 is known as the *Accept–Reject method*, which is usually stated in the slightly modified, but equivalent form. (See Problem 2.28 for extensions.)

Algorithm A.4 –Accept–Reject Method–

1. Generate $X \sim g$, $U \sim \mathcal{U}_{[0,1]}$;
2. Accept $Y = X$ if $U \leq f(X)/Mg(X)$; [A.4]
3. Return to 1. otherwise.

In cases where f and g are normalized so they are both probability densities, the constant M is necessarily larger than 1. Therefore, the size of M, and thus the efficiency of [A.4], becomes a function of how closely g can imitate f, especially in the tails of the distribution. Note that for f/g to remain bounded, it is necessary for g to have tails thicker than those of f. It is therefore impossible for instance to use [A.4] to simulate a Cauchy distribution f using a normal distribution g; however, the reverse works quite well. (See Problem 2.34.) Interestingly enough, the opposite case when g/f is bounded can also be processed by a tailored Markov chain Monte Carlo algorithm derived from Doukhan et al. (1994) (see Problems 7.5 and 7.6).

A limited optimization of the Accept–Reject algorithm is possible by choosing the instrumental density g in a parametric family, and then determining the value of the parameter which minimizes the bound M. A similar comparison between two parametric families is much more delicate since it is then necessary to take into account the computation time of one generation from g in [A.4]. In fact, pushing the reasoning to the limit, if $g = f$

and if we simulate $X \sim f$ by numerical inversion of the distribution function, we formally achieve the minimal bound $M = 1$, but this does not guarantee that we have an efficient algorithm, as can be seen in the case of the normal distribution.

Example 2.18. Normals from double exponentials. Consider generating a $\mathcal{N}(0,1)$ by [A.4] using a double-exponential distribution $\mathcal{L}(\alpha)$, with density $g(x|\alpha) = (\alpha/2)\exp(-\alpha|x|)$. It is then straightforward to show that

$$\frac{f(x)}{g(x|\alpha)} \leq \sqrt{2/\pi}\alpha^{-1}e^{\alpha^2/2}$$

and that the minimum of this bound (in α) is attained for $\alpha = 1$. The probability of acceptance is then $\sqrt{\pi/2e} = .76$, which shows that to produce one normal random variable, this Accept–Reject algorithm requires on the average $1/.76 \approx 1.3$ uniform variables, to be compared with the fixed single uniform required by the Box–Muller algorithm. ||

A real advantage of the Accept–Reject algorithm is illustrated in the following example.

Example 2.19. Gamma Accept–Reject We saw in Example 2.7 that if $\alpha \in \mathbb{N}$, the Gamma distribution $\mathcal{G}a(\alpha, \beta)$ can be represented as the sum of α exponential random variables $\epsilon_i \sim \mathcal{E}xp(\beta)$, which are very easy to simulate, since $\epsilon_i = -\log(U_i)/\beta$, with $U_i \sim \mathcal{U}([0,1])$. In more general cases (for example when $\alpha \notin \mathbb{N}$), this representation does not hold.

A possible approach is to use the Accept–Reject algorithm with instrumental distribution $\mathcal{G}a(a,b)$, with $a = [\alpha]$ $(\alpha \geq 1)$. (Without loss of generality, suppose $\beta = 1$.) The ratio f/g is $b^{-a}x^{\alpha-a}\exp\{-(1-b)x\}$, up to a normalizing constant, yielding the bound

$$M = b^{-a}\left(\frac{\alpha - a}{(1-b)e}\right)^{\alpha-a}$$

for $b < 1$. Since the maximum of $b^{-a}(1-b)^{\alpha-a}$ is attained at $b = a/\alpha$, the optimal choice of b for simulating $\mathcal{G}a(\alpha,1)$ is $b = a/\alpha$, which gives the same mean for $\mathcal{G}a(\alpha,1)$ and $\mathcal{G}a(a,b)$. (See Problem 2.31.) ||

It may also happen that the complexity of the optimization is very expensive in terms of analysis or of computing time. In the first case, the construction of the optimal algorithm should still be undertaken when the algorithm is to be subjected to intensive use. In the second case, it is most often preferable to explore the use of another family of instrumental distributions g.

Example 2.20. Truncated normal distributions. *Truncated normal distributions* appear in many contexts, such as in the discussion after Example 1.5. When constraints $x \geq \underline{\mu}$ produce densities proportional to

$$e^{-(x-\mu)^2/2\sigma^2}\, \mathbb{I}_{x \geq \underline{\mu}}$$

for a bound $\underline{\mu}$ large compared with μ, there are alternatives which are far superior to the naïve method in which a $\mathcal{N}(\mu, \sigma^2)$ distribution is simulated until the generated value is larger than $\underline{\mu}$. (This approach requires an average number of $1/\Phi((\mu - \underline{\mu})/\sigma)$ simulations from $\mathcal{N}(\mu, \sigma^2)$ for one acceptance.) Consider, without loss of generality, the case $\mu = 0$ and $\sigma = 1$. A potential instrumental distribution is the translated exponential distribution, $\mathcal{E}xp(\alpha, \underline{\mu})$, with density

$$g_\alpha(z) = \alpha e^{-\alpha(z-\underline{\mu})}\, \mathbb{I}_{z \geq \underline{\mu}}\,.$$

The ratio $f/g_\alpha(z) = e^{-\alpha(z-\underline{\mu})}\, e^{-z^2/2}$ is then bounded by $\exp(\alpha^2/2 - \alpha\underline{\mu})$ if $\alpha > \underline{\mu}$ and by $\exp(-\underline{\mu}^2/2)$ otherwise. The corresponding (upper) bound is

$$\begin{cases} 1/\alpha\, \exp(\alpha^2/2 - \alpha\underline{\mu}) & \text{if } \alpha > \underline{\mu}, \\ 1/\alpha\, \exp(-\underline{\mu}^2/2) & \text{otherwise.} \end{cases}$$

The first expression is minimized by

$$\alpha^* = \underline{\mu} + \frac{1}{2}\sqrt{\underline{\mu}^2 + 4}\,, \tag{2.11}$$

whereas $\tilde{\alpha} = \underline{\mu}$ minimizes the second bound. The optimal choice of α is therefore (2.11), which requires the computation of the square root of $\underline{\mu}^2 + 4$. Robert (1995b) proposes a similar algorithm for the case where the normal distribution is restricted to the interval $[\underline{\mu}, \overline{\mu}]$. For some values of $[\underline{\mu}, \overline{\mu}]$, the optimal algorithm is associated with a value of α, which is a solution to an implicit equation. (See also Geweke 1991 for a similar resolution of this simulation problem and Marsaglia 1964 for an earlier solution.) ‖

One criticism of the Accept–Reject algorithm is that it generates "useless" simulations when rejecting. We will see in Chapter 3 how the method of importance sampling (Section 3.3) can be used to bypass this problem and also how both methods can be compared.

2.4 Envelope Accept–Reject Methods

2.4.1 The Squeeze Principle

In numerous settings, the distribution associated with the density f is difficult to simulate because of the complexity of the function f itself, which may require substantial computing time at each evaluation. In the setup of Example 1.9 for instance, if a Bayesian approach is taken with θ distributed (a posteriori) as

$$\prod_{i=1}^{n} \left[1 + \frac{(x_i - \theta)^2}{p\sigma^2}\right]^{-\frac{p+1}{2}} , \tag{2.12}$$

where σ is known, each single evaluation of $\pi(\theta|x)$ involves the computation of n terms in the product. It turns out that an acceleration of the simulation of densities such as (2.12) can be accomplished by an algorithm that is "one step beyond" the Accept–Reject algorithm. This algorithm is an *envelope* algorithm and relies on the evaluation of a simpler function g_l which bounds the target density f from below. The algorithm is based on the following extension of Corollary 2.17 (see Problems 2.35 and 2.36).

Lemma 2.21. *If there exist a density g_m, a function g_l and a constant M such that*

$$g_l(x) \leq f(x) \leq M g_m(x) ,$$

then the algorithm

Algorithm A.5 –Envelope Accept–Reject–

1. Generate $X \sim g_m(x)$, $U \sim \mathcal{U}_{[0,1]}$;
2. Accept X if $U \leq g_l(X)/Mg_m(X)$; [A.5]
3. otherwise, accept X if $U \leq f(X)/Mg_m(X)$

produces random variables that are distributed according to f.

By the construction of a lower envelope on f, based on the function g_l, the number of evaluations of f is potentially decreased by a factor

$$\frac{1}{M} \int g_l(x)\, dx,$$

which is the probability that f is not evaluated. This method is called the *squeeze principle* by Marsaglia (1977) and the ARS algorithm [A.7] in Section 2.4 is based on it. A possible way of deriving the bounds g_l and Mg_m is to use a Taylor expansion of $f(x)$.

Example 2.22. Lower bound for normal generation. It follows from the Taylor series expansion of $\exp(-x^2/2)$ that $\exp(-x^2/2) \geq 1 - (x^2/2)$, and hence

$$\left(1 - \frac{x^2}{2}\right) \leq f(x),$$

which can be interpreted as a lower bound for the simulation of $\mathcal{N}(0,1)$. This bound is obviously useless when $|X| < \sqrt{2}$, an event which occurs with probability 0.61 for $X \sim \mathcal{C}(0,1)$. ||

Example 2.23. Poisson variables from logistic variables. As indicated in Example 2.9, the simulation of the Poisson distribution $\mathcal{P}(\lambda)$ using a Poisson process and exponential variables can be rather inefficient. Here, we describe a simpler alternative of Atkinson (1979), who uses the relationship between the Poisson $\mathcal{P}(\lambda)$ distribution and the *logistic distribution*. The logistic distribution has density and distribution function

$$f(x) = \frac{1}{\beta} \frac{\exp\{-(x-\alpha)/\beta\}}{[1+\exp\{-(x-\alpha)/\beta\}]^2} \quad \text{and} \quad F(x) = \frac{1}{1+\exp\{-(x-\alpha)/\beta\}}$$

and is therefore analytically invertible.

To better relate the continuous and discrete distributions, we consider $N = \lfloor x + 0.5 \rfloor$, the integer part of $x + 0.5$. Also, the range of the logistic distribution is $(-\infty, \infty)$, but to better match it with the Poisson, we restrict the range to $[-1/2, \infty)$. Thus, the random variable N has distribution function

$$P(N = n) = \frac{1}{1+e^{-(n+0.5-\alpha)/\beta}} - \frac{1}{1+e^{-(n-0.5-\alpha)/\beta}}$$

if $x > 1/2$ and

$$P(N = n) = \left(\frac{1}{1+e^{-(n+0.5-\alpha)/\beta}} - \frac{1}{1+e^{-(n-0.5-\alpha)/\beta}} \right) \frac{1+e^{-(0.5+\alpha)/\beta}}{e^{-(0.5+\alpha)/\beta}}$$

if $-1/2 < x \leq 1/2$ and the ratio of the densities is

$$\lambda^n / P(N = n)\, e^{\lambda}\, n! \,. \tag{2.13}$$

Although it is difficult to compute a bound on (2.13) and, hence, to optimize it in (α, β), Atkinson (1979) proposed the choice $\alpha = \lambda$ and $\beta = \pi/\sqrt{3\lambda}$. This identifies the two first moments of X with those of $\mathcal{P}(\lambda)$. For this choice of α and β, analytic optimization of the bound on (2.13) remains impossible, but numerical maximization and interpolation yields the bound $c = 0.767 - 3.36/\lambda$. The resulting algorithm is then

Algorithm A.6 –Atkinson's Poisson Simulation–

0. Define $\beta = \pi/\sqrt{3\lambda}$, $\alpha = \lambda\beta$ and $k = \log c - \lambda - \log\beta$;
1. Generate $U_1 \sim \mathcal{U}_{[0,1]}$ and calculate

$$x = \{\alpha - \log\{(1-u_1)/u_1\}\}/\beta$$

   until $X > -0.5$;
2. Define $N = \lfloor X + 0.5 \rfloor$ and generate $U_2 \sim \mathcal{U}_{[0,1]}$;
3. Accept $N \sim \mathcal{P}(\lambda)$ if [A.6]

$$\alpha - \beta x + \log\left(u_2/\{1+\exp(\alpha-\beta x)\}^2\right) \leq k + N\log\lambda - \log N! \,.$$

Although the resulting simulation is exact, this algorithm is based on a number of approximations, both through the choice of (α, β) and in the computation of the majorization bounds and the density ratios. Moreover, note that it requires the computation of factorials, $N!$, which may be quite time-consuming. Therefore, although [A.6] usually has a reasonable efficiency, more complex algorithms such as those of Devroye (1985) may be preferable. ‖

2.4.2 Log-Concave Densities

The particular case of *log-concave densities* (that is, densities whose logarithm is concave) allows the construction of a generic algorithm that can be quite efficient.

Example 2.24. Log-concave densities. Recall the exponential family (1.9)

$$f(x) = h(x)\, e^{\theta \cdot x - \psi(\theta)}, \qquad \theta, x \in \mathbb{R}^k.$$

This density is log-concave if

$$\frac{\partial^2}{\partial x^2} \log f(x) = \frac{\partial^2}{\partial x^2} \log h(x) = \frac{h(x)h''(x) - [h'(x)]^2}{h^2(x)} < 0,$$

which will often be the case for the exponential family. For example, if $X \sim \mathcal{N}(\theta, 1)$, then $h(x) \propto \exp\{-x^2/2\}$ and $\partial^2 \log h(x)/\partial x^2 = -1$. See Problems 2.40–2.42 for properties and examples of log-concave densities. ‖

Devroye (1985) describes some algorithms that take advantage of the log-concavity of the density, but here we present a universal method. The algorithm, which was proposed by Gilks (1992) and Gilks and Wild (1992), is based on the construction of an envelope and the derivation of a corresponding Accept–Reject algorithm. The method is called *adaptive rejection sampling* (ARS) and it provides a sequential evaluation of lower and upper envelopes of the density f when $h = \log f$ is concave.

Let $\mathcal{S}_n$ be a set of points $x_i, i = 0, 1, \ldots, n+1$, in the support of f such that $h(x_i) = \log f(x_i)$ is known up to the same constant. Given the concavity of h, the line $L_{i,i+1}$ through $(x_i, h(x_i))$ and $(x_{i+1}, h(x_{i+1}))$ is below the graph of h in $[x_i, x_{i+1}]$ and is above this graph outside this interval (see Figure 2.5). For $x \in [x_i, x_{i+1}]$, if we define

$$\overline{h}_n(x) = \min\{L_{i-1,i}(x), L_{i+1,i+2}(x)\} \quad \text{and} \quad \underline{h}_n(x) = L_{i,i+1}(x)\,,$$

the envelopes are

$$\underline{h}_n(x) \le h(x) \le \overline{h}_n(x) \tag{2.14}$$

uniformly on the support of f. (We define

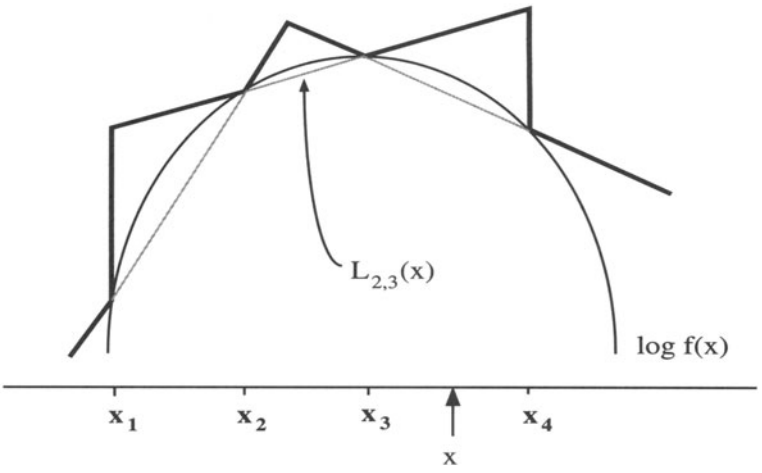

Fig. 2.5. Lower and upper envelopes of $h(x) = \log f(x)$, f a log-concave density (*Source:* Gilks et al. 1995).

$$\underline{h}_n(x) = -\infty \quad \text{and} \quad \overline{h}_n(x) = \min(L_{0,1}(x), L_{n,n+1}(x))$$

on $[x_0, x_{n+1}]^c$.) Therefore, for $\underline{f}_n(x) = \exp \underline{h}_n(x)$ and $\overline{f}_n(x) = \exp \overline{h}_n(x)$, (2.14) implies that

$$\underline{f}_n(x) \le f(x) \le \overline{f}_n(x) = \varpi_n \, g_n(x) \,,$$

where ϖ_n is the normalized constant of f_n; that is, g_n is a density. The ARS algorithm to generate an observation from f is thus

Algorithm A.7 –ARS Algorithm–

0. Initialize n and $\mathcal{S}_n$.
1. Generate $X \sim g_n(x)$, $U \sim \mathcal{U}_{[0,1]}$.
2. If $U \le \underline{f}_n(X)/\varpi_n \, g_n(X)$, accept X; [A.7]
   otherwise, if $U \le f(X)/\varpi_n \, g_n(X)$, accept X
   and update $\mathcal{S}_n$ to $\mathcal{S}_{n+1} = \mathcal{S}_n \cup \{X\}$.

An interesting feature of this algorithm is that the set $\mathcal{S}_n$ is only updated when $f(x)$ has been previously computed. As the algorithm produces variables $X \sim f(x)$, the two envelopes $\underline{f}_n$ and $\overline{f}_n$ become increasingly accurate and, therefore, we progressively reduce the number of evaluations of f. Note that in the initialization of $\mathcal{S}_n$, a necessary condition is that $\varpi_n < +\infty$ (i.e., that g_n is actually a probability density). To achieve this requirement, $L_{0,1}$ needs to have positive slope if the support of f is not bounded on the left and $L_{n,n+1}$ needs to have a negative slope if the support of f is not bounded on the right. (See Problem 2.39 for more on simulation from g_n.)

The ARS algorithm is not optimal in the sense that it is often possible to devise a better specialized algorithm for a given log-concave density. However, although Gilks and Wild (1992) do not provide theoretical evaluations of simulation speeds, they mention reasonable performances in the cases they consider. Note that, in contrast to the previous algorithms, the function g_n is updated during the iterations and, therefore, the average computation time for one generation from f decreases with n. This feature makes the comparison with other approaches quite delicate.

The major advantage of [A.7] compared with alternatives is its *universality*. For densities f that are only known through their functional form, the ARS algorithm yields an automatic Accept–Reject algorithm that only requires checking f for log-concavity. Moreover, the set of log-concave densities is wide; see Problems 2.40 and 2.41. The ARS algorithm thus allows for the generation of samples from distributions that are rarely simulated, without requiring the development of case-specific Accept–Reject algorithms.

Example 2.25. Capture–recapture models. In a heterogeneous *capture–recapture model* (see Seber 1983, 1992 or Borchers et al. 2002), animals are captured at time i with probability p_i, the size N of the population being unknown. The corresponding likelihood is therefore

$$L(p_1,\ldots,p_I|N,n_1,\ldots,n_I) = \frac{N!}{(N-r)!}\prod_{i=1}^{I} p_i^{n_i}(1-p_i)^{N-n_i},$$

where I is the number of captures, n_i is the number of captured animals during the ith capture, and r is the total number of different captured animals. If N is a priori distributed as a $\mathcal{P}(\lambda)$ variable and the p_i's are from a *normal logistic* model,

$$\alpha_i = \log\left(\frac{p_i}{1-p_i}\right) \sim \mathcal{N}(\mu_i,\sigma^2),$$

as in George and Robert (1992), the posterior distribution satisfies

$$\pi(\alpha_i|N,n_1,\ldots,n_I) \propto \exp\left\{\alpha_i n_i - \frac{1}{2\sigma^2}(\alpha_i-\mu_i)^2\right\} \Big/ (1+e^{\alpha_i})^N .$$

If this conditional distribution must be simulated (for reasons which will be made clearer in Chapters 9 and 10), the ARS algorithm can be implemented. In fact, the log of the posterior distribution

$$\alpha_i n_i - \frac{1}{2\sigma^2}(\alpha_i-\mu_i)^2 - N\,\log(1+e^{\alpha_i}) \tag{2.15}$$

is concave in α_i, as can be shown by computing the second derivative (see also Problem 2.42).

As an illustration, consider the dataset $(n_1,\ldots,n_{11}) = (32,20,8,5,1,2,0,2,1,1,0)$ which describes the number of recoveries over the years 1957–1968 of

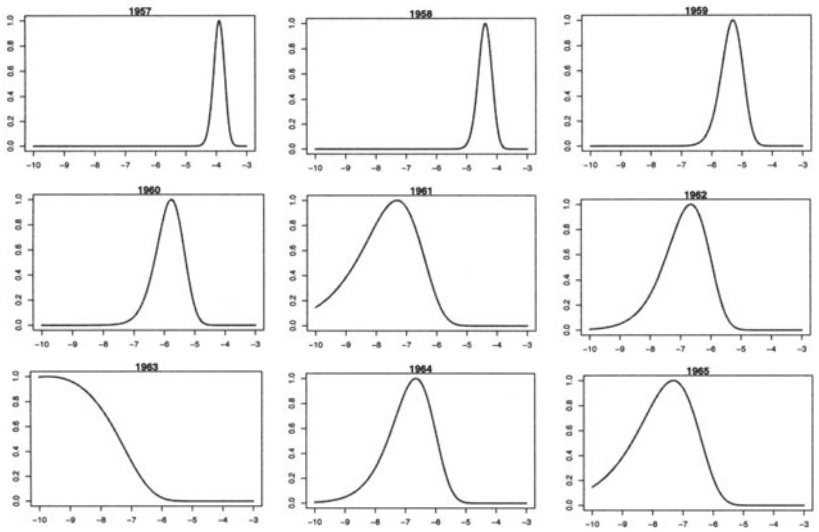

Fig. 2.6. Posterior distributions of the capture log-odds ratios for the Northern Pintail duck dataset of Johnson and Hoeting (2003) for the years 1957–1965.

$N = 1612$ Northern Pintail ducks banded in 1956, as reported in Johnson and Hoeting (2003). Figure 2.6 provides the corresponding posterior distributions for the first 9 α_i's. The ARS algorithm can then be used independently for each of these distributions. For instance, if we take the year 1960, the starting points in $\mathcal{S}$ can be $-10, -6$ and -3. The set $\mathcal{S}$ then gets updated along iterations as in Algorithm [A.7], which provides a correct simulation from the posterior distributions of the α_i's, as illustrated in Figure 2.7 for the year 1960. ||

The above example also illustrates that checking for log-concavity of a Bayesian posterior distribution is straightforward, as $\log \pi(\theta|x) = \log \pi(\theta) + \log f(x|\theta) + c$, where c is a constant (in θ). This implies that the log-concavity of $\pi(\theta)$ and of $f(x|\theta)$ (in θ) are sufficient to conclude the log-concavity of $\pi(\theta|x)$.

Example 2.26. Poisson regression. Consider a sample $(Y_1, x_1), \ldots, (Y_n, x_n)$ of integer-valued data Y_i with explanatory variable x_i, where Y_i and x_i are connected via a Poisson distribution,

$$Y_i|x_i \sim \mathcal{P}(\exp\{a + bx_i\}) .$$

If the prior distribution of (a, b) is a normal distribution $\mathcal{N}(0, \sigma^2) \times \mathcal{N}(0, \tau^2)$, the posterior distribution of (a, b) is given by

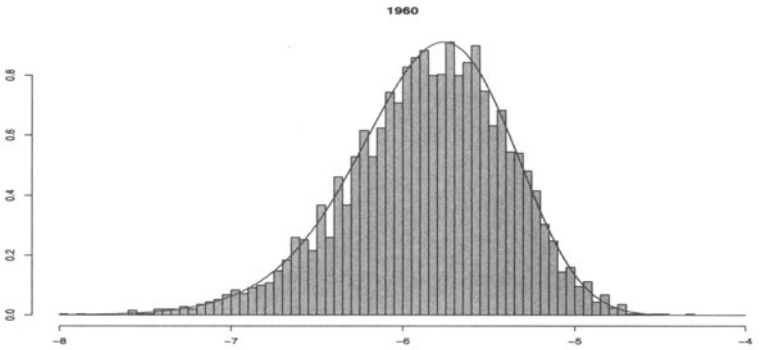

Fig. 2.7. Histogram of an ARS sample of 5000 points and corresponding posterior distribution of the log-odds ratio α_{1960}.

$$\pi(a,b|\mathbf{x},\mathbf{y}) \propto \exp\left\{a\sum_i y_i + b\sum_i y_i x_i - e^a \sum_i e^{x_i b}\right\} e^{-a^2/2\sigma^2}\, e^{-b^2/2\tau^2}.$$

We will see in Chapter 9 that it is often of interest to simulate successively the (full) conditional distributions $\pi(a,|\mathbf{x},\mathbf{y},b)$ and $\pi(b|\mathbf{x},\mathbf{y},a)$. Since

$$\log \pi(a|\mathbf{x},\mathbf{y},b) = a\sum_i y_i - e^a \sum_i e^{x_i b} - a^2/2\sigma^2\,,$$
$$\log \pi(b|\mathbf{x},\mathbf{y},a) = b\sum_i y_i x_i - e^a \sum_i e^{x_i b} - b^2/2\tau^2\,,$$

and

$$\frac{\partial^2}{\partial a^2} \log \pi(a|\mathbf{x},\mathbf{y},b) = -\sum_i e^{x_i b}\, e^a - \sigma^{-2} < 0\,,$$
$$\frac{\partial^2}{\partial b^2} \log \pi(b|\mathbf{x},\mathbf{y},a) = -e^a \sum_i x_i^2\, e^{x_i b} - \tau^{-2} < 0\,,$$

the ARS algorithm directly applies for both conditional distributions.

As an illustration, consider the data in Table 2.1. This rather famous data set gives the deaths in the Prussian Army due to kicks from horses, gathered by von Bortkiewicz (1898). A question of interest is whether there is a trend in the deaths over time. For illustration here, we show how to generate the conditional distribution of the intercept, $\pi(a|\mathbf{x},\mathbf{y},b)$, since the generation of the other conditional is quite similar.

Before implementing the ARS algorithm, we note two simplifying things. One, if $f(x)$ is easy to compute (as in this example), there is really no need to construct $\underline{f}_n(x)$, and we just skip that step in Algorithm [A.7]. Second, we do not need to construct the function g_n, we only need to know how to simulate

Table 2.1. Data from the 19^{th} century study by Bortkiewicz (1898) of deaths in the Prussian army due to horse kicks. The data are the number of deaths in fourteen army corps from 1875 to 1894.

Year	75	76	77	78	79	80	81	82	83	84
Deaths	3	5	7	9	10	18	6	14	11	9
Year	85	86	87	88	89	90	91	92	93	94
Deaths	5	11	15	6	11	17	12	15	8	4

from it. To do this, we only need to compute the area of each segment above the intervals $[x_i, x_{i+1}]$.

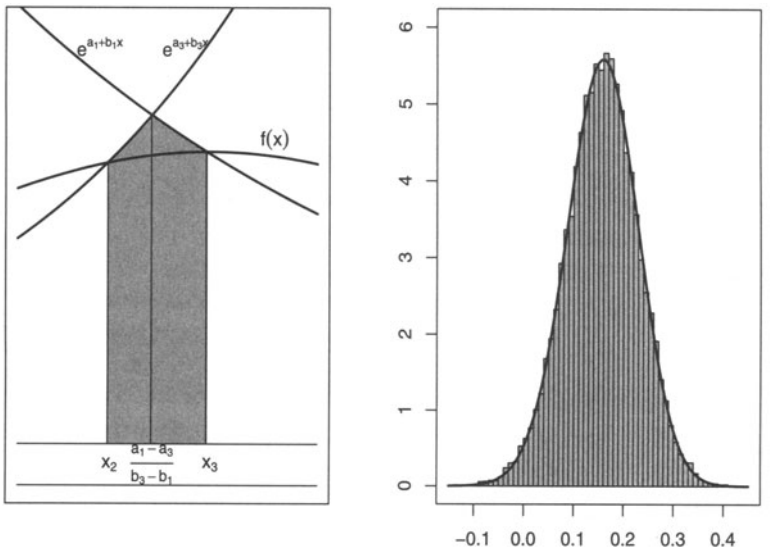

Fig. 2.8. Left panel is the area of integration for the weight of the interval $[x_2, x_3]$. The right panel is the histogram and density of the sample from g_n, with $b = .025$ and $\sigma^2 = 5$.

The left panel of Figure 2.8 shows the region of the support of $f(x)$ between x_2 and x_3, with the grey shaded area proportional to the probability of selecting the region $[x_2, x_3]$. If we denote by $a_i + b_i x$ the line through $(x_i, h(x_i))$ and $(x_{i+1}, h(x_{i+1}))$, then the area of the grey region of Figure 2.8 is

$$\begin{aligned} \omega_2 &= \int_{x_2}^{\frac{a_1-a_3}{b_3-b_1}} e^{a_1+b_1 x}\, dx + \int_{\frac{a_1-a_3}{b_3-b_1}}^{x_3} e^{a_3+b_3 x}\, dx \\ &= \frac{e^{a_1}}{b_1}\left[e^{\frac{a_1-a_3}{b_3-b_1}b1} - e^{b_1 x_2}\right] + \frac{e^{a_3}}{b_3}\left[e^{b_3 x_3} - e^{\frac{a_1-a_3}{b_3-b_1}b_3}\right]. \end{aligned} \tag{2.16}$$

Thus, to sample from g_n we choose a region $[x_i, x_{i+1}]$ proportional to ω_i, generate $U \sim \mathcal{U}_{[0,1]}$ and then take

$$X = x_i + U(x_{i+1} - x_i.)$$

The right panel of Figure 2.8 shows the good agreement between the histogram and density of g_n. (See Problems 2.37 and 2.38 for generating the g_n corresponding to $\pi(b|\mathbf{x}, \mathbf{y}, a)$, and Problems 9.7 and 9.8 for full Gibbs samplers.) ||

2.5 Problems

2.1 Check the uniform random number generator on your computer:
(a) Generate 1,000 uniform random variables and make a histogram
(b) Generate uniform random variables $(X_1, \ldots, X_n)$ and plot the pairs (X_i, X_{i+1}) to check for autocorrelation.

2.2 (a) Generate a binomial $\mathcal{B}in(n, p)$ random variable with $n = 25$ and $p = .2$. Make a histogram and compare it to the binomial mass function, and to the R binomial generator.
(b) Generate 5,000 *logarithmic series* random variables with mass function

$$P(X = x) = \frac{-(1-p)^x}{x \log p}, \quad x = 1, 2, \ldots \quad 0 < p < 1.$$

Make a histogram and plot the mass function.

2.3 In each case generate the random variables and compare to the density function
(a) Normal random variables using a Cauchy candidate in Accept–Reject;
(b) Gamma $\mathcal{G}a(4.3, 6.2)$ random variables using a Gamma Ga(4, 7);
(c) Truncated normal: Standard normal truncated to $(2, \infty)$.

2.4 The arcsine distribution was discussed in Example 2.2.
(a) Show that the *arcsine distribution*, with density $f(x) = 1/\pi\sqrt{x(1-x)}$, is invariant under the transform $y = 1 - x$, that is, $f(x) = f(y)$.
(b) Show that the uniform distribution $\mathcal{U}_{[0,1]}$ is invariant under the "tent" transform,

$$D(x) = \begin{cases} 2x & \text{if } x \leq 1/2 \\ 2(1-x) & \text{if } x > 1/2. \end{cases}$$

(c) As in Example 2.2, use both the arcsine and "tent" distributions in the dynamic system $X_{n+1} = D(X_n)$ to generate 100 uniform random variables. Check the properties with marginal histograms, and plots of the successive iterates.
(d) The tent distribution can have disastrous behavior. Given the finite representation of real numbers in the computer, show that the sequence (X_n) will converge to a fixed value, as the tent function progressively eliminates the last decimals of X_n. (For example, examine what happens when the sequence starts at a value of the form $1/2^n$.)

2.5 For each of the following distributions, calculate the explicit form of the distribution function and show how to implement its generation starting from a uniform random variable: (*a*) exponential; (*b*) double exponential; (*c*) Weibull; (*d*) Pareto; (*e*) Cauchy; (*f*) extreme value; (*g*) arcsine.

2.6 Referring to Example 2.7:

(a) Show that if $U \sim \mathcal{U}_{[0,1]}$, then $X = -\log U/\lambda \sim \mathcal{E}xp(\lambda)$.

(b) Verify the distributions in (2.2).

(c) Show how to generate an $\mathcal{F}_{m,n}$ random variable, where both m and n are even integers.

(d) Show that if $U \sim \mathcal{U}_{[0,1]}$, then $X = \log \frac{u}{1-u}$ is a Logistic$(0, 1)$ random variable. Show also how to generate a Logistic(μ, β) random variable.

2.7 Establish the properties of the *Box–Muller algorithm* of Example 2.8. If U_1 and U_2 are iid $\mathcal{U}_{[0,1]}$, show that:

(a) The transforms

$$X_1 = \sqrt{-2\log(U_1)}\,\cos(2\pi U_2)\,, \qquad X_2 = \sqrt{-2\log(U_1)}\,\sin(2\pi U_2)\,,$$

are iid $\mathcal{N}(0, 1)$.

(b) The polar coordinates are distributed as

$$r^2 = X_1^2 + X_2^2 \sim \chi_2^2\,,$$
$$\theta = \arctan\frac{X_1}{X_2} \sim \mathcal{U}[0, 2\pi].$$

(c) Establish that $\exp(-r^2/2) \sim \mathcal{U}[0, 1]$, and so r^2 and θ can be simulated directly.

2.8 (Continuation of Problem 2.7)

(a) Show that an alternate version of the Box–Muller algorithm is

Algorithm A.8 –Box–Muller (2)–

1. Generate [*A*.8]

$$U_1, U_2 \sim \mathcal{U}([-1, 1])$$

until $S = U_1^2 + U_2^2 \leq 1$.

2. Define $Z = \sqrt{-2\log(S)/S}$ and take

$$X_1 = Z\,U_1, \qquad X_2 = Z\,U_2.$$

(*Hint:* Show that (U_1, U_2) is uniform on the unit sphere and that X_1 and X_2 are independent.)

(b) Give the average number of generations in `1.` and compare with the original Box–Muller algorithm [*A*.3] on a small experiment.

(c) Examine the effect of not constraining (U_1, U_2) to the unit circle.

2.9 Show that the following version of the Box–Muller algorithm produces one normal variable and compare the execution time with both versions [*A*.3] and [*A*.8]:

Algorithm A.9 –Box–Muller (3)–

1. Generate
$$Y_1, Y_2 \sim \mathcal{E}xp(1)$$
until $Y_2 > (1-Y_1)^2/2$. [A.9]
2. Generate $U \sim \mathcal{U}([0,1])$ and take
$$X = \begin{cases} Y_1 & \text{if } U < 0.5 \\ -Y_1 & \text{if } U > 0.5. \end{cases}$$

2.10 Examine the properties of an algorithm to simulate $\mathcal{N}(0,1)$ random variables based on the Central Limit Theorem, which takes the appropriately adjusted mean of a sequence of uniforms $U_1, \ldots, U_n$ for $n = 12$, $n = 48$ and $n = 96$. Consider, in particular, the moments, ranges, and tail probability calculations based on the generated variables.

2.11 For the generation of a Cauchy random variable, compare the inversion method with one based on the generation of the normal pair of the polar Box–Muller method (Problem 2.8).

(a) Show that, if X_1 and X_2 are iid normal, $Y = X_1/X_2$ is distributed as a Cauchy random variable.

(b) Show that the Cauchy distribution function is $F(x) = \tan^{-1}(x)/\pi$, so the inversion method is easily implemented.

(c) Is one of the two algorithms superior?

2.12 Use the algorithm of Example 2.10 to generate the following random variables. In each case make a histogram and compare it to the mass function, and to the generator in your computer.

(a) Binomials and Poisson distributions;

(b) The hypergeometric distribution;

(c) The *logarithmic series* distribution. A random variable X has a logarithmic series distribution with parameter p if
$$P(X = x) = \frac{-(1-p)^x}{x \log p}, \quad x = 1, 2, \ldots, \quad 0 < p < 1.$$

(d) Referring to part (a), for different parameter values, compare the algorithms there with those of Problems 2.13 and 2.16.

2.13 Referring to Example 2.9.

(a) Show that if $N \sim \mathcal{P}(\lambda)$ and $X_i \sim \mathcal{E}xp(\lambda)$, $i \in \mathbb{N}^*$, independent, then
$$P_\lambda(N = k) = P_\lambda(X_1 + \cdots + X_k \leq 1 < X_1 + \cdots + X_{k+1}) .$$

(b) Use the results of part (a) to justify that the following algorithm simulates a Poisson $\mathcal{P}(\lambda)$ random variable:

Algorithm A.10 –Poisson simulation–

$p = 1$, $N = 0$, $c = e^{-\lambda}$.
1. Repeat [A.10]
$N = N + 1$
generate U_i
update $p = pU_i$
until $p < c$.
2. Take $X = N - 1$.

(*Hint:* For part (*a*), integrate the Gamma density by parts.)

2.14 There are (at least) two ways to establish (2.4) of Example 2.12:
(a) Make the transformation $x = yu^{1/\alpha}$, $w = y$, and integrate out w.
(b) Make the transformation $x = yz$, $w = z$, and integrate out w.

2.15 In connection with Example 2.16, for a Beta distribution $\mathcal{B}e(\alpha, \beta)$, find the maximum of the $\mathcal{B}e(\alpha, \beta)$ density.

2.16 Establish the validity of Knuth (1981) $\mathcal{B}(n, p)$ generator:

Algorithm A.11 –Binomial–

Define $k = n$, $\theta = p$ and $x = 0$.
1. Repeat $i = [1 + k\theta]$
$V \sim \mathcal{B}e(i, k + 1 - i)$ [A.11]
if $\theta > V$, $\theta = \theta/V$ and $k = i - 1$;
otherwise, $x = x + i$, $\theta = (\theta - V)/(1 - V)$ and $k = k - i$
until $k \leq K$.
2. For $i = 1, 2, \ldots, k$,
generate U_i
if $U_i < p$, $x = x + 1$.
3. Take x.

2.17 Establish the claims of Example 2.11: If $U_1, \ldots, U_n$ is an iid sample from $\mathcal{U}_{[0,1]}$ and $U_{(1)} \leq \cdots \leq U_{(n)}$ are the corresponding order statistics, show that
(a) $U_{(i)} \sim \mathcal{B}e(i, n - i + 1)$;
(b) $(U_{(i_1)}, U_{(i_2)} - U_{(i_1)}, \ldots, U_{(i_k)} - U_{(i_{k-1})}, 1 - U_{(i_k)}) \sim \mathcal{D}(i_1, i_2 - i_1, \ldots, n - i_k + 1)$;
(c) If U and V are iid $\mathcal{U}_{[0,1]}$, the distribution of

$$\frac{U^{1/\alpha}}{U^{1/\alpha} + V^{1/\beta}},$$

conditional on $U^{1/\alpha} + V^{1/\beta} \leq 1$, is the $\mathcal{B}e(\alpha, \beta)$ distribution.
(d) Show that the order statistics can be directly generated via the *Renyi representation* $u_{(i)} = \sum_{j=1}^{i} \nu_j / \sum_{j=1}^{n} \nu_j$, where the ν_j's are iid $\mathcal{E}xp(1)$.

2.18 For the generation of a Student's t distribution, $\mathcal{T}(\nu, 0, 1)$, Kinderman et al. (1977) provide an alternative to the generation of a normal random variable and a chi squared random variable.

Algorithm A.12 –Student's t–

1. Generate $U_1, U_2 \sim \mathcal{U}([0, 1])$.
2. If $U_1 < 0.5$, $X = 1/(4U_1 - 1)$ and $V = X^{-2}U_2$;
otherwise, $X = 4U_1 - 3$ and $V = U_2$. [A.12]
3. If $V < 1 - (|X|/2)$ or $V < (1 + (X^2/\nu))^{-(\nu+1)/2}$, take X;
otherwise, repeat.

Validate this algorithm and compare it with the algorithm of Example 2.13.

2.19 For $\alpha \in [0, 1]$, show that the algorithm

Algorithm A.13

Generate
$U \sim \mathcal{U}([0, 1])$ [A.13]
until $U < \alpha$.

produces a simulation from $\mathcal{U}([0,\alpha])$. Compare it with the transform αU, $U \sim \mathcal{U}(0,1)$ for values of α close to 0 and close to 1.

2.20 In each case generate the random variables and compare the histogram to the density function

(*a*) Normal random variables using a Cauchy candidate in Accept–Reject

(*b*) Gamma(4.3, 6.2) random variables using a Gamma(4, 7).

(*c*) Truncated normal: Standard normal truncated to $(2,\infty)$

2.21 An efficient algorithm for the simulation of Gamma $\mathcal{G}a(\alpha,1)$ distributions is based on *Burr's distribution*, a distribution with density

$$g(x) = \lambda\mu \frac{x^{\lambda-1}}{(\mu+x^{\lambda})^2}, \qquad x > 0 .$$

It has been developed by Cheng (1977) and Cheng and Feast (1979). (See Devroye 1985.) For $\alpha > 1$, it is

Algorithm A.14 –Cheng and Feast's Gamma–

Define $c_1 = \alpha - 1$, $c_2 = (\alpha - (1/6\alpha))/c_1$, $c_3 = 2/c_1$, $c_4 = 1 + c_3$, and $c_5 = 1/\sqrt{\alpha}$.

1. Repeat
   generate U_1, U_2
   take $U_1 = U_2 + c_5(1 - 1.86U_1)$ if $\alpha > 2.5$
   until $0 < U_1 < 1$. [A.14]
2. $W = c_2 U_2/U_1$.
3. If $c_3 U_1 + W + W^{-1} \leq c_4$ or $c_3 \log U_1 - \log W + W \leq 1$,
   take $c_1 W$;
   otherwise, repeat.

(a) Show g is a density.

(b) Show that this algorithm produces variables generated from $\mathcal{G}a(\alpha,1)$.

2.22 Ahrens and Dieter (1974) propose the following algorithm to generate a Gamma $\mathcal{G}a(\alpha,1)$ distribution:

Algorithm A.15 –Ahrens and Dieter's Gamma–

1. Generate U_0, U_1.
2. If $U_0 > e/(e+\alpha)$, $x = -\log\{(\alpha+e)(1-U_0)/\alpha e\}$ and $y = x^{\alpha-1}$;
   otherwise, $x = \{(\alpha+e)U_0/e\}^{1/\alpha}$ and $y = e^{-x}$. [A.15]
3. If $U_1 < y$, take x;
   otherwise, repeat.

Show that this algorithm produces variables generated from $\mathcal{G}a(\alpha,1)$. Compare with Problem 2.21.

2.23 To generate the Beta distribution $\mathcal{B}e(\alpha,\beta)$ we can use the following representation:

(a) Show that, if $Y_1 \sim \mathcal{G}a(\alpha,1)$, $Y_2 \sim \mathcal{G}a(\beta,1)$, then

$$X = \frac{Y_1}{Y_1+Y_2} \sim \mathcal{B}e(\alpha,\beta) .$$

(b) Use part (a) to construct an algorithm to generate a Beta random variable.

(c) Compare this algorithm with the method given in Problem 2.17 for different values of (α, β).

(d) Compare this algorithm with an Accept–Reject algorithm based on (i) the uniform distribution; (ii) the truncated normal distribution (when $\alpha \geq 1$ and $\beta \geq 1$).

(*Note:* See Schmeiser and Shalaby 1980 for an alternative Accept–Reject algorithm to generate Beta rv's.)

2.24 (a) Show that Student's t density can be written in the form (2.6), where $h_1(x|y)$ is the density of $\mathcal{N}(0, \nu/y)$ and $h_2(y)$ is the density of χ^2_ν.

(b) Show that Fisher's $\mathcal{F}_{m,\nu}$ density can be written in the form (2.6), with $h_1(x|y)$ the density of $\mathcal{G}a(m/2, \nu/m)$ and $h_2(y)$ the density of χ^2_ν.

2.25 The noncentral chi squared distribution, $\chi^2_p(\lambda)$, can be defined by a mixture representation (2.6), where $h_1(x|K)$ is the density of χ^2_{p+2K} and $h_2(k)$ is the density of $\mathcal{P}(\lambda/2)$.

(a) Show that it can also be expressed as the sum of a χ^2_{p-1} random variable and of the square of a standard normal variable.

(b) Compare the two algorithms which can be derived from these representations.

(c) Discuss whether a direct approach via an Accept–Reject algorithm is at all feasible.

2.26 (Walker 1997) Show that the Weibull distribution, $\mathcal{W}e(\alpha, \beta)$, with density

$$f(x|\alpha, \beta) = \beta\alpha x^{\alpha-1} \exp\left(-\beta x^\alpha\right),$$

can be represented as a mixture of $X \sim \mathcal{B}e(\alpha, \omega^{1/\alpha})$ by $\omega \sim \mathcal{G}a(2, \beta)$. Examine whether this representation is helpful from a simulation point of view.

2.27 An application of the mixture representation can be used to establish the following result (see Note 2.6.3):

Lemma 2.27. *If*

$$f(x) = \frac{f_1(x) - \varepsilon f_2(x)}{1 - \varepsilon},$$

where f_1 and f_2 are probability densities such that $f_1(x) \geq \varepsilon f_2(x)$, the algorithm

Generate

$$(X, U) \sim f_1(x)\, \mathbb{I}_{[0,1]}(u)$$

until $U \geq \varepsilon f_2(X)/f_1(X)$.

produces a variable X distributed according to f.

(a) Show that the distribution of X satisfies

$$P(X \leq x_0) = \int_{-\infty}^{x_0} \left(1 - \frac{\varepsilon f_2(x)}{f_1(x)}\right) f_1(x)\, dx \sum_{i=0}^{\infty} \varepsilon^i .$$

(b) Evaluate the integral in (a) to complete the proof.

2.28 (a) Demonstrate the equivalence of Corollary 2.17 and the Accept–Reject algorithm.

(b) Generalize Corollary 2.17 to the multivariate case. That is, for $\mathbf{X} = (X_1, X_2, \ldots, X_p) \sim f(x_1, x_2, \ldots, x_p) = f(\mathbf{x})$, formulate a joint distribution $(\mathbf{X}, U) \sim \mathbb{I}(0 < u < f(\mathbf{x}))$, and show how to generate a sample from $f(\mathbf{x})$ based on uniform random variables.

2.29 (a) Referring to (2.10), if $\pi(\theta|x)$ is the target density in an Accept–Reject algorithm, and $\pi(\theta)$ is the candidate density, show that the bound M can be taken to be the likelihood function evaluated at the MLE.

(b) For estimating a normal mean, a robust prior is the Cauchy. For $X \sim N(\theta, 1)$, $\theta \sim \text{Cauchy}(0, 1)$, the posterior distribution is

$$\pi(\theta|x) \propto \frac{1}{\pi(1+\theta^2)} \frac{1}{2\pi} e^{-(x-\theta)^2/2}.$$

Use the Accept–Reject algorithm, with a Cauchy candidate, to generate a sample from the posterior distribution.

(*Note:* See Problem 3.19 and Smith and Gelfand 1992.)

2.30 For the Accept–Reject algorithm [A.4], with f and g properly normalized,

(a) Show that the probability of accepting a random variable is

$$P\left(U < \frac{f(X)}{Mg(X)}\right) = \frac{1}{M}.$$

(b) Show that $M \geq 1$.

(c) Let N be the number of trials until the kth random variable is accepted. Show that, for the normalized densities, N has the negative binomial distribution $\mathcal{N}eg(k, p)$, where $p = 1/M$. Deduce that the expected number of trials until k random variables are obtained is kM.

(d) Show that the bound M does not have to be tight; that is, there may be $M' < M$ such that $f(x) \leq M'g(x)$. Give an example where it makes sense to use M instead of M'.

(e) When the bound M is too tight (i.e., when $f(x) > Mg(x)$ on a non-negligible part of the support of f), show that the algorithm [*A*.4] does not produce a generation from f. Give the resulting distribution.

(f) When the bound is not tight, show that there is a way, using Lemma 2.27, to recycle part of the rejected random variables. (*Note:* See Casella and Robert 1998 for details.)

2.31 For the Accept–Reject algorithm of the $\mathcal{G}a(n, 1)$ distribution, based on the $\mathcal{E}xp(\lambda)$ distribution, determine the optimal value of λ.

2.32 This problem looks at a generalization of Example 2.19.

(a) If the target distribution of an Accept–Reject algorithm is the Gamma distribution $\mathcal{G}a(\alpha, \beta)$, where $\alpha \geq 1$ is not necessarily an integer, show that the instrumental distribution $\mathcal{G}a(a, b)$ is associated with the ratio

$$\frac{f(x)}{g(x)} = \frac{\Gamma(a)}{\Gamma(\alpha)} \frac{\beta^\alpha}{b^a} x^{\alpha-a} e^{-(\beta-b)x}.$$

(b) Why do we need $a < \alpha$ and $b < \beta$?

(c) For $a = \lfloor \alpha \rfloor$, show that the bound is maximized (in x) at $x = (\alpha - a)/(\beta - b)$.

(d) For $a = \lfloor \alpha \rfloor$, find the optimal choice of b.

(e) Compare with $a' = \lfloor \alpha \rfloor - 1$, when $\alpha > 2$.

2.33 The right-truncated Gamma distribution $\mathcal{TG}(a, b, t)$ is defined as the restriction of the Gamma distribution $\mathcal{G}a(a, b)$ to the interval $(0, t)$.

(a) Show that we can consider $t = 1$ without loss of generality.

(b) Give the density f of $\mathcal{TG}(a, b, 1)$ and show that it can be expressed as the following mixture of Beta $\mathcal{B}e(a, k + 1)$ densities:

$$f(x) = \sum_{k=0}^{\infty} \frac{b^a e^{-b}}{\gamma(a, b)} \frac{b^k}{k!} x^{a-1}(1 - x)^k,$$

where $\gamma(a, b) = \int_0^1 x^{a-1} e^{-x}$.

(c) If f is replaced with g_n which is the series truncated at term $k = n$, show that the acceptance probability of the Accept–Reject algorithm based on (g_n, f) is

$$\frac{1 - \dfrac{\gamma(n + 1, b)}{n!}}{1 - \dfrac{\gamma(a + n + 1, b)\Gamma(a)}{\Gamma(a + n + 1)\gamma(a, b)}}.$$

(d) Evaluate this probability for different values of (a, b).

(e) Give an Accept–Reject algorithm based on the pair (g_n, f) and a computable bound. (*Note:* See Philippe 1997c for a complete resolution of the problem.)

2.34 Let $f(x) = \exp(-x^2/2)$ and $g(x) = 1/(1 + x^2)$, densities of the normal and Cauchy distributions, respectively (ignoring the normalization constants).

(a) Show that the ratio

$$\frac{f(x)}{g(x)} = (1 + x^2)\, e^{-x^2/2} \leq 2/\sqrt{e},$$

which is attained at $x = \pm 1$.

(b) Show that for the normalized densities, the probability of acceptance is $\sqrt{e/2\pi} = 0.66.$, which implies that, on the average, one out of every three simulated Cauchy variables is rejected. Show that the mean number of trials to success is $1/.66 = 1.52$.

(c) Replacing g by a Cauchy density with scale parameter σ,

$$g_\sigma(x) = 1/\{\pi\sigma(1 + x^2/\sigma^2)\},$$

show that the bound on f/g_σ is $2\sigma^{-1} \exp\{\sigma^2/2 - 1\}$ and is minimized by $\sigma^2 = 1$. (This shows that $\mathcal{C}(0, 1)$ is the best choice among the Cauchy distributions for simulating a $\mathcal{N}(0, 1)$ distribution.)

2.35 There is a direct generalization of Corollary 2.17 that allows the proposal density to change at each iteration.

Algorithm A.16 –Generalized Accept–Reject–

At iteration i $(i \geq 1)$

1. Generate $X_i \sim g_i$ and $U_i \sim \mathcal{U}([0, 1])$, independently.
2. If $U_i \leq \epsilon_i f(X_i)/g_i(X_i)$, accept $X_i \sim f$;
3. otherwise, move to iteration $i + 1$.

(a) Let Z denote the random variable that is output by this algorithm. Show that Z has the cdf

$$P(Z \le z) = \sum_{i=1}^{\infty} \epsilon_i \prod_{j=1}^{i-1}(1-\epsilon_j) \int_{-\infty}^{z} f(x)dx\,.$$

(b) Show that

$$\sum_{i=1}^{\infty} \epsilon_i \prod_{j=1}^{i-1}(1-\epsilon_j) = 1 \text{ if and only if } \sum_{i=1}^{\infty} \log(1-\epsilon_i) \text{ diverges}\,.$$

Deduce that we have a valid algorithm if the second condition is satisfied.

(c) Give examples of sequences ϵ_i that satisfy, and do not satisfy, the requirement of part (*b*).

2.36 (a) Prove the validity of the ARS Algorithm [A.7], without the envelope step, by applying Algorithm [A.16].

(b) Prove the validity of the ARS Algorithm [A.7], with the envelope step, directly. Note that

$$P(X \le x|\text{Accept}) = P\left(X \le x \middle| \left\{U < \frac{g_\ell}{Mg_m} \text{ or } U < \frac{f}{Mg_m}\right\}\right)$$

and

$$\left\{U < \frac{g_\ell}{Mg_m} \text{ or } U < \frac{f}{Mg_m}\right\} = \left\{U < \frac{g_\ell}{Mg_m}\right\} \cup \left\{\frac{g_\ell}{Mg_m} < U < \frac{f}{Mg_m}\right\},$$

which are disjoint.

2.37 Based on the discussion in Example 2.26, write an alternate algorithm to Algorithm [A.17] that does not require the calculation of the density g_n.

2.38 The histogram and density of Figure 2.8 give the candidate g_n for $\pi(a|\mathbf{x},\mathbf{y},b)$, the conditional density of the intercept a in $\log\lambda = a+bt$, where we set $b = .025$ and $\sigma^2 = 5$. Produce the same picture for the slope, b, when we set the intercept $a = .15$ and $\tau^2 = 5$.

2.39 Step 1 of [A.7] relies on simulations from g_n. Show that we can write

$$g_n = \varpi_n^{-1}\left\{\sum_{i=0}^{r_n} e^{\alpha_i x+\beta_i}\,\mathbb{I}_{[x_i,x_{i+1}]}(x) + e^{\alpha_{-1}x+\beta_{-1}}\mathbb{I}_{[-\infty,x_0]}(x)\right.$$
$$\left. e^{\alpha_{r_n+1}x+\beta_{r_n+1}}\,\mathbb{I}_{[x_{n+1},+\infty]}(x)\right\},$$

where $y = \alpha_i x + \beta_i$ is the equation of the segment of line corresponding to g_n on $[x_i, x_{i+1}]$, r_n denotes the number of segments, and

$$\varpi_n = \int_{-\infty}^{x_0} e^{\alpha_{-1}x+\beta_{-1}}dx + \sum_{i=0}^{n}\int_{x_i}^{x_{i+1}} e^{\alpha_i x+\beta_i}dx + \int_{x_{n+1}}^{+\infty} e^{\alpha_{r_n+1}x+\beta_{r_n+1}}dx$$
$$= \frac{e^{\alpha_{-1}x_0+\beta_{-1}}}{\alpha_{-1}} + \sum_{i=0}^{n} e^{\beta_i}\,\frac{e^{\alpha_i x_{i+1}}-e^{\alpha_i x_i}}{\alpha_i} - \frac{e^{\alpha_{r_n+1}x_{n+1}}}{\alpha_{r_n+1}}\,,$$

when supp $f = \mathbb{R}$.

Verify that this representation as a sequence validates the following algorithm for simulation from g_n:

Algorithm A.17 –Supplemental ARS Algorithm–

1. Select the interval $[x_i, x_{i+1}]$ with probability

$$e^{\beta_i} \frac{e^{\alpha_i x_{i+1}} - e^{\alpha_i x_i}}{\varpi_n \alpha_i}. \qquad [A.17]$$

2. Generate $U \sim \mathcal{U}_{[0,1]}$ and take

$$X = \alpha_i^{-1} \log[e^{\alpha_i x_i} + U(e^{\alpha_i x_{i+1}} - e^{\alpha_i x_i})].$$

Note that the segment $\alpha_i + \beta_i x$ is not the same as the line $a_i + b_i x$ used in (2.16)

2.40 As mentioned in Section 2.4, many densities are log-concave.

(a) Show that the so-called *natural* exponential family,

$$dP_\theta(x) = \exp\{x \cdot \theta - \psi(\theta)\} d\nu(x)$$

is log-concave.

(b) Show that the logistic distribution of (2.23) is log-concave.

(c) Show that the *Gumbel* distribution

$$f(x) = \frac{k^k}{(k-1)!} \exp\left\{-kx - ke^{-x}\right\}, \quad k \in \mathbb{N}^*,$$

is log-concave (Gumbel 1958).

(d) Show that the *generalized inverse Gaussian* distribution,

$$f(x) \propto x^\alpha e^{-\beta x - \alpha/x}, \qquad x > 0,\ \alpha > 0,\ \beta > 0,$$

is log-concave.

2.41 (George et al. 1993) For the natural exponential family, the conjugate prior measure is defined as

$$d\pi(\theta | x_0, n_0) \propto \exp\{x_0 \cdot \theta - n_0 \psi(\theta)\} d\theta,$$

with $n_0 > 0$. (See Brown 1986, Chapter 1, for properties of exponential families.)

(a) Show that

$$\varphi(x_0, n_0) = \log \int_\Theta \exp\{x_0 \cdot \theta - n_0 \psi(\theta)\} d\theta$$

is convex.

(b) Show that the so-called *conjugate likelihood distribution*

$$L(x_0, n_0 | \theta_1, \ldots, \theta_p) \propto \exp\left\{x_0 \cdot \sum_{i=1}^p \theta_i - n_0 \sum_{i=1}^p \psi(\theta) - p\varphi(x_0, n_0)\right\}$$

is log-concave in (x_0, n_0).

(c) Deduce that the ARS algorithm applies in hierarchical Bayesian models with conjugate priors on the natural parameters and log-concave hyperpriors on (x_0, n_0).

(d) Apply the ARS algorithm to the case

$$X_i | \theta_i \sim \mathcal{P}(\theta_i t_i), \qquad \theta_i \sim \mathcal{G}a(\alpha, \beta), \qquad i = 1, \ldots, n,$$

with fixed α and $\beta \sim \mathcal{G}a(0.1, 1)$.

2.42 In connection with Example 2.25,
(a) Show that a sum of log-concave functions is a log-concave function.
(b) Deduce that (2.15) is log-concave.

2.43 (Casella and Berger 2001, Section 8.3) This problem examines the relationship of the property of log-concavity with other desirable properties of density functions.
(a) The property of *monotone likelihood ratio* is very important in the construction of hypothesis tests, and in many other theoretical investigations. A family of pdfs or pmfs $\{g(t|\theta) : \theta \in \Theta\}$ for a univariate random variable T with real-valued parameter θ has a *monotone likelihood ratio* (MLR) if, for every $\theta_2 > \theta_1$, $g(t|\theta_2)/g(t|\theta_1)$ is a monotone (nonincreasing or nondecreasing) function of t on $\{t : g(t|\theta_1) > 0 \text{ or } g(t|\theta_2) > 0\}$. Note that $c/0$ is defined as ∞ if $0 < c$.
Show that if a density is log-concave, it has a monotone likelihood ratio.
(b) Let $f(x)$ be a pdf and let a be a number such that, if $a \geq x \geq y$ then $f(a) \geq f(x) \geq f(y)$ and, if $a \leq x \leq y$ then $f(a) \geq f(x) \geq f(y)$. Such a pdf is called *unimodal* with a *mode* equal to a.
Show that if a density is log-concave, it is unimodal.

2.44 This problem will look into one of the failings of congruential generators, the production of parallel lines of output. Consider a congruential generator $D(x) = ax \bmod 1$, that is, the output is the fractional part of ax.
(a) For $k = 1, 2, \ldots, 333$, plot the pairs $(k*0.003, D(k*0.003))$ for $a = 5, 20, 50$. What can you conclude about the parallel lines?
(b) Show that each line has slope a and the lines repeat at intervals of $1/a$ (hence, larger values of a will increase the number of lines). (*Hint:* Let $x = \frac{i}{a} + \delta$, for $i = 1, \ldots, a$ and $0 < \delta < \frac{1}{a}$. For this x, show that $D(x) = a\delta$, regardless of the value of i.)

2.6 Notes

2.6.1 The Kiss Generator

Although this book is not formally concerned with the generation of uniform random variables (as we start from the assumption that we have an endless supply of such variables), it is good to understand the basic workings and algorithms that are used to generate these variables. In this note we describe the way in which uniform pseudo-random numbers are generated, and give a particularly good algorithm.

To keep our presentation simple, rather than give a catalog of random number generators, we only give details for a single generator, the Kiss algorithm of Marsaglia and Zaman (1993). For details on other random number generators, the books of Knuth (1981), Rubinstein (1981), Ripley (1987), and Fishman (1996) are excellent sources.

As we have remarked before, the finite representation of real numbers in a computer can radically modify the behavior of a dynamic system. Preferred generators are those that take into account the specifics of this representation and provide a uniform sequence. It is important to note that such a sequence does not really take values in the interval $[0, 1]$ but rather on the integers $\{0, 1, \ldots, M\}$, where M is the largest integer accepted by the computer. One manner of characterizing the performance of these integer generators is through the notion of *period*.

Definition 2.28. The *period*, T_0, of a generator is the smallest integer T such that $u_{i+T} = u_i$ for every i; that is, such that D^T is equal to the identity function.

The period is a very important parameter, having direct impact on the usefulness of a random number generator. If the number of needed generations exceeds the period of a generator, there may be noncontrollable artifacts in the sequence (cyclic phenomena, false orderings, etc.). Unfortunately, a generator of the form $X_{n+1} = f(X_n)$ has a period no greater than $M+1$, for obvious reasons. In order to overcome this bound, a generator must utilize several sequences X_n^i simultaneously (which is a characteristic of Kiss) or must involve $X_{n-1}, X_{n-2}, \ldots$ in addition to X_n, or must use other methods such as *start-up tables*, that is, using an auxiliary table of random digits to restart the generator.

Kiss simultaneously uses two generation techniques, namely *congruential* generation and *shift register* generation.

Definition 2.29. A *congruential generator* on $\{0, 1, \ldots, M\}$ is defined by the function

$$D(x) = (ax + b) \bmod (M + 1).$$

The period and, more generally, the performance of congruential generators depend heavily on the choice of (a, b) (see Ripley 1987). When transforming the above generator into a generator on $[0, 1]$, with $\tilde{D}(x) = (ax+b)/(M+1) \bmod 1$, the graph of D should range throughout $[0, 1]^2$, and a choice of the constant $a \notin \mathbb{Q}$ would yield a "recovery " of $[0, 1]^2$; that is, an infinite sequence of points should fill the space.

Although ideal, the choice of an irrational a is impossible (since a needs be specified with a finite number of digits). With a rational, a congruential generator will produce pairs $(x_n, D(x_n))$ that lie on parallel lines. Figure 2.9 illustrates this phenomenon for $a = 69069$, representing the sequence $(3k\ 10^{-4}, D(3k))$ for $k = 1, 2, \ldots, 333$. It is thus important to select a in such a way as to maximize the number of parallel segments in $[0, 1]^2$ (see Problem 2.44).

Most commercial generators use congruential methods, with perhaps the most disastrous choice of (a, b) being that of the old (and notorious) procedure *RANDU* (see Ripley 1987). Even when the choice of (a, b) assures the acceptance of the generator by standard tests, nonuniform behavior will be observed in the last digits of the real numbers produced by this method, due to round-up errors.

The second technique employed by Kiss is based on the (theoretical) independence between the k binary components of $X_n \sim \mathcal{U}_{\{0,1,\ldots,M\}}$ (where $M = 2^k - 1$) and is called a *shift register* generator.

Definition 2.30. For a given $k \times k$ matrix T, whose entries are either 0 or 1, the associated *shift register generator* is given by the transformation

$$x_{n+1} = Tx_n,$$

where x_n is represented as a vector of binary coordinates e_{ni}, that is to say,

$$x_n = \sum_{i=0}^{k-1} e_{ni} 2^i,$$

with e_{ni} equal to 0 or 1.

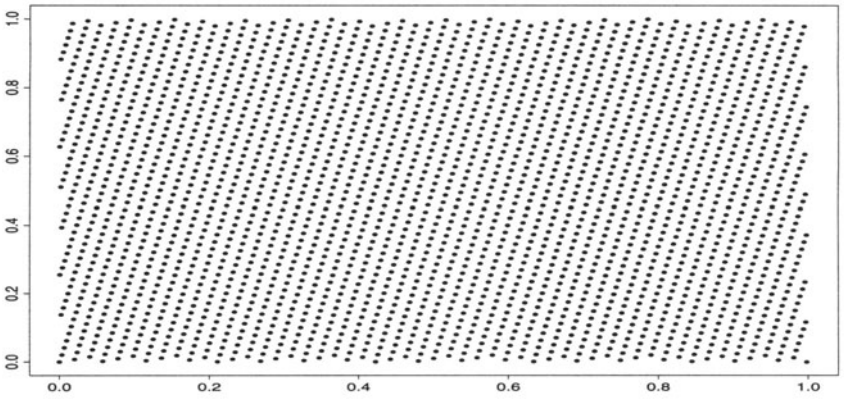

Fig. 2.9. Representation of the line $y = 69069x \mod 1$ by uniform sampling with sampling step $3\,\dot{1}0^{-4}$.

This second class of generators is motivated by both the internal (computer-dependent) representation of numbers as sequences of *bits* and the speed of manipulation of these elementary algebraic operations. Since the computation of Tx_n is done modulo 2, each addition is then the equivalent of a logical OR. Moreover, as the matrix T only contains 0 and 1 entries, multiplication by T amounts to shifting the content of coordinates, which gives the technique its name.

For instance, if the ith line of T contains a 1 in the ith and jth positions uniquely, the ith coordinate of x_{n+1}, will be obtained by

$$\begin{aligned} e_{(n+1)i} &= (e_{ni} + e_{nj}) \mod 2 \\ &= e_{ni} \vee e_{nj} - e_{ni} \wedge e_{nj} \,, \end{aligned}$$

where $a \wedge b = \min(a, b)$ and $a \vee b = \max(a, b)$. This is a comparison of the ith coordinate of x_n and the coordinate corresponding to a *shift* of $(j - i)$. There also exist sufficient conditions on T for the associated generator to have period 2^k (see Ripley 1987).

The generators used by Kiss are based on the matrices

$$T_L = \begin{pmatrix} 1 & 1 & & \\ & \ddots & & 0 \\ & & \ddots & 1 \\ 0 & & & 1 \end{pmatrix} \quad \text{and} \quad T_R = \begin{pmatrix} 1 & & & \\ 1 & \ddots & 0 & \\ & & \ddots & \\ & 0 & 1 & 1 \end{pmatrix},$$

whose entries are 1 on the main diagonal and on the first upper diagonal and first lower diagonal, respectively, the other elements being 0. They are related to the right and left shift matrices,

$$\begin{aligned} R(e_1, \ldots, e_k)^t &= (0, e_1, \ldots, e_{k-1})^t, \\ L(e_1, \ldots, e_k)^t &= (e_2, e_3, \ldots, e_k, 0)^t, \end{aligned}$$

since $T_R = (I + R)$ and $T_L = (I + L)$.

To generate a sequence of integers $X_1, X_2, \ldots$, the Kiss algorithm generates three sequences of integers. First, the algorithm uses a congruential generator to obtain

$$I_{n+1} = (69069 \times I_n + 23606797) \pmod{2^{32}} ,$$

and then two shift register generators of the form,

$$\begin{aligned} J_{n+1} &= (I + L^{15})(I + R^{17})\, J_n \pmod{2^{32}} , \\ K_{n+1} &= (I + L^{13})(I + R^{18})\, K_n \pmod{2^{31}} . \end{aligned}$$

These are then combined to produce

$$X_{n+1} = (I_{n+1} + J_{n+1} + K_{n+1}) \mod 2^{32} .$$

Formally, this algorithm is not of the type specified in Definition 2.1, since it uses three parallel chains of integers. However, this feature yields advantages over algorithms based on a single dynamic system $X_{n+1} = f(X_n)$ since the period of Kiss is of order 2^{95}, which is almost $(2^{32})^3$. In fact, the (usual) congruential generator I_n has a maximal period of 2^{32}, the generator of K_n has a period of $2^{31} - 1$ and that of J_n a period of $2^{32} - 2^{21} - 2^{11} + 1$ for almost all initial values J_0 (see Marsaglia and Zaman 1993 for more details). The Kiss generator has been successfully tested on the different criteria of Die Hard, including tests on random subsets of *bits*. Figure 2.10 presents plots of (X_n, X_{n+1}), (X_n, X_{n+2}), (X_n, X_{n+5}) and (X_n, X_{n+10}) for $n = 1, \ldots, 5000$, where the sequence (X_n) has been generated by Kiss, without exhibiting any nonuniform feature on the square $[0, 1]^2$. A version of this algorithm in the programming language C is given below.

Algorithm A.18 –The Kiss Algorithm–

```
long int kiss (i,j,k)
unsigned long *i,*j,*k
{
*j = *j ∧ (*j << 17);                                    [A.18]
*k = (*k ∧ (*k << 18)) & 0X7FFFFFFF ;
return  ((*i = 69069 * (*i) + 23606797) +
(*j ∧ = (*j >> 15)) + (*k ∧ = (*k >> 13)) ;
}
```

(See Marsaglia and Zaman 1993 for a Fortran version of Kiss). Note that some care must be exercised in the use of this program as a generator on $[0, 1]$, since it implies dividing by the largest integer available on the computer and may sometimes result in uniform generation on $[-1, 1]$.

2.6.2 Quasi-Monte Carlo Methods

Quasi-Monte Carlo methods were proposed in the 1950s to overcome some drawbacks of regular Monte Carlo methods by replacing probabilistic bounds on the errors with deterministic bounds. The idea at the core of quasi-Monte Carlo methods is to substitute the randomly (or pseudo-randomly) generated (uniform $[0, 1]$) sequences

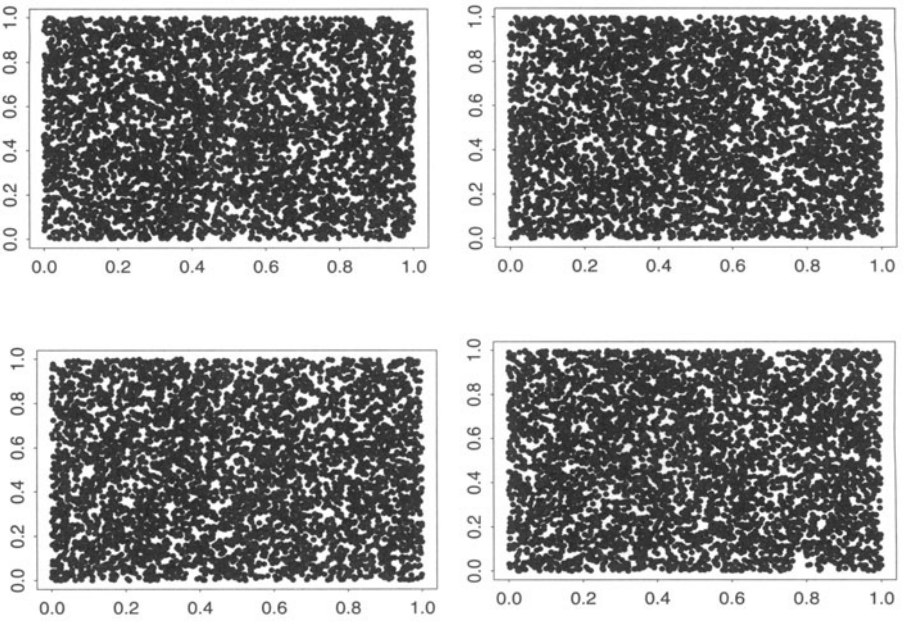

Fig. 2.10. Plots of pairs (X_t, X_{t+1}), (X_t, X_{t+2}), (X_t, X_{t+5}) and (X_t, X_{t+10}) for a sample of 5000 generations from Kiss.

used in regular Monte Carlo methods with a deterministic sequence (x_n) on $[0,1]$ in order to minimize the so-called *divergence*,

$$D(x_1, \ldots, x_n) = \sup_{0 \le u \le 1} \left| \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{[0,u]}(x_i) - u \right| .$$

This is also the *Kolmogorov–Smirnov distance* between the empirical cdf and that of the uniform distribution, used in nonparametric tests. For fixed n, the solution is obviously $x_i = (2i-1)/2n$ in dimension 1, but the goal here is to get a *low-discrepancy* sequence (x_n) which provides small values of $D(x_1, \ldots, x_n)$ for all n's (i.e., such that $x_1, \ldots, x_{n-1}$ do not depend on n) and can thus be updated sequentially.

As shown in Niederreiter (1992), there exist such sequences, which ensure a divergence rate of order $\mathcal{O}(n^{-1} \log(n)^{d-1})$, where d is the dimension of the integration space.[6] Since, for any function h defined on $[0,1]$, it can be shown that the divergence is related to the overall approximation error by

$$\left| \frac{1}{n} \sum_{i=1}^{n} h(x_i) - \int_0^1 h(x)dx \right| \le V(h) D(x_1, \ldots, x_n) \tag{2.17}$$

(see Niederreiter 1992), where $V(h)$ is the total variation of h,

[6] The notation $\mathcal{O}(1/n)$ denotes a function that satisfies $0 < \lim_{n \to \infty} n\mathcal{O}(1/n) < \infty$.

$$V(h) = \lim_{N\to\infty} \sup_{0\le x_1\cdots x_{N-1}\le 1} \sum_{j=1}^{N} |h(x_j) - h(x_{j-1})|\,,$$

with $x_0 = 0$ and $x_N = 1$, the gain over standard Monte Carlo methods can be substantial since standard methods lead to order $\mathcal{O}(n^{-1/2})$ errors (see Section 4.3). The advantage over standard integration techniques such as Riemann sums is also important when the dimension d increases since the latter are of order $n^{4/d}$ (see Yakowitz et al. 1978).

The true comparison with regular Monte Carlo methods is, however, more delicate than a simple assessment of the order of convergence. Construction of these sequences, although independent from h, can be quite involved (see Fang and Wang 1994 for examples), even though they only need to be computed *once*. More importantly, the construction requires that the functions to be integrated have bounded support, which can be a hindrance in practice because the choice of the transformation to $[0,1]^d$ is crucial for the efficiency of the method. See Niederreiter (1992) for extensions in optimization setups.

2.6.3 Mixture Representations

Mixture representations, such as those used in Examples 2.13 and 2.14, can be extended (theoretically, at least) to many other distributions. For instance, a random variable X (and its associated distribution) is called *infinitely divisible* if for every n there exist iid random variables $X'_1, \cdots, X'_n$ such that $X \sim X'_1 + \cdots + X'_n$ (see Feller 1971, Section XVII.3 or Billingsley 1995, Section 28). It turns out that most distributions that are infinitely divisible can be represented as mixtures of Poisson distributions, the noncentral $\chi^2_p(\lambda)$ distribution being a particular case of this phenomenon. (However, this theoretical representation does not necessarily guarantee that infinitely divisible distributions are always easy to simulate.)

We also note that if the finite mixture

$$\sum_{i=1}^{k} p_i\, f_i(x)$$

can result in a decomposition of $f(x)$ into simple components (for instance, uniform distributions on intervals) and a last residual term with a small weight, the following approximation applies: We can use a trapezoidal approximation of f on intervals $[a_i, b_i]$, the weight p_i being of the order of $\int_{a_i}^{b_i} f(x)dx$. Devroye (1985) details the applications of this method in the case where f is a polynomial on $[0,1]$.