Chapter 8
Simulation from the Tail of the Univariate and Multivariate Normal Distribution

Zdravko Botev and Pierre L'Ecuyer

8.1 Introduction

We consider the problem of simulating a standard normal random variable X, conditional on $a \leq X \leq b$, where $a < b$ are real numbers, and at least one of them is finite. We are particularly interested in the situation where the interval (a, b) is far in one of the tails, that is, we assume that $a \gg 0$ (the case where $b \ll 0$ is covered by symmetry). We do not consider the case where $a \leq 0 \leq b$, as it can be handled easily via standard methods, which do not always work well in the tail case $a \gg 0$. Moreover, if we insist on using inversion, the standard inversion methods break down when we are far in the tail. Inversion is preferable to a rejection method (in general) in various simulation applications, for example to maintain synchronization and monotonicity when comparing systems with common random numbers, for derivative estimation and optimization, when using quasi-Monte Carlo methods, etc. [6, 12–15]. For this reason, a good inversion method is needed, even if rejection is faster. We examine both rejection and inversion methods in this paper.

These problems occur in particular for the estimation of certain Bayesian regression models and for exact simulation from these models; see [4, 7] and the references given there. The simulation from the Bayesian posterior requires repeated draws from a standard normal distribution truncated to different intervals, often far in the tail. Note that to generate X from a more general normal distribution with

Z. Botev (✉)
UNSW Sydney, Sydney, NSW, Australia
e-mail: botev@unsw.edu.au

P. L'Ecuyer
Université de Montréal, Montréal, QC, Canada
Inria - Rennes, Rennes, France
e-mail: lecuyer@iro.umontreal.ca

A. Puliafito, K. S. Trivedi (eds.), *Systems Modeling: Methodologies and Tools*, EAI/Springer Innovations in Communication and Computing,
https://doi.org/10.1007/978-3-319-92378-9_8

mean μ and variance σ^2 truncated to an interval (a', b'), it suffices to apply a simple linear transformation to recover the standard normal problem studied here.

This paper has three main contributions.

1. *Comparison amongst univariate methods.* The first contribution is to review and compare the speed and efficiency of some of the most popular methods [7, 8, 11, 18, 22, 24] for the tail of the univariate normal distribution. These methods are designed to be efficient when $a \gg 0$ and $b = \infty$ (or $a = -\infty$ and $b \ll 0$ by symmetry), and are not necessarily efficient when the interval $[a, b]$ contains 0. We find that these methods may be adapted in principle to a finite interval $[a, b]$, but they may become inefficient when the interval $[a, b]$ is narrow. We also find that the largely ignored (or forgotten) method of Marsaglia [19] is typically more efficient than the widely used accept–reject methods of Geweke [9] and Robert [22].
2. *Accurate inversion for univariate truncated normal.* All of the methods cited above are rejection methods and we found no reliable inversion method for an interval far in the tail (say, for $a > 38$; see Sect. 8.2). Our second contribution is to propose a new accurate inversion method for arbitrarily large a. Our inversion algorithm is based on a numerically stable implementation of the solution of a nonlinear equation via Newton's method.
3. *Rejection method for multivariate truncated normal.* Our third contribution is to propose a simple rejection method in the multivariate setting, where we wish to simulate a vector $\boldsymbol{X}$ with mean zero and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$, conditional on $\boldsymbol{X} \geq \boldsymbol{a}$ (the inequality is componentwise). We find that, under some conditions, the proposed method can yield an acceptance probability that approaches unity as we move deeper into the tail region.

Simulation methods for exact simulation from multivariate normal distributions conditional on a general rectangular region, $\boldsymbol{a} \leq \boldsymbol{X} \leq \boldsymbol{b}$, were developed recently in [3, 4, 6]. But for sampling in the tail, the proposed sampler in this paper has two advantages compared to the samplers in these previous works. First, it is much simpler to implement and faster, because it is specifically designed for the tail of the multivariate normal. Second, the theoretical results in [3] do not apply when the target pdf is the most general tail density (see (8.9) in Sect. 8.3), but they do apply for our proposal in this paper. On the downside, the price one pays for these two advantages is that the proposed sampler works only in the extreme tail setting ($[\boldsymbol{a}, \infty]$ with $\boldsymbol{a} \gg \boldsymbol{0}$), whereas the methods in [3, 4, 6] work in more general non-tail settings ($[\boldsymbol{a}, \boldsymbol{b}]$ which may contain $\boldsymbol{0}$).

This chapter is an expanded version of the conference paper [5]. The results of Sect. 8.3 are new while those of Sect. 8.2 were contained in [5].

8.2 Simulation from the Tail of the Univariate Normal

In this section, we use ϕ to denote the density of the standard normal distribution (with mean 0 and variance 1), Φ for its cumulative distribution function (cdf), $\overline{\Phi}$ for the complementary cdf, and Φ^{-1} for the inverse cdf defined as $\Phi^{-1}(u) = \min\{x \in \mathbb{R} \mid \Phi(x) \geq u\}$. Thus, if $X \sim \mathsf{N}(0, 1)$, $\Phi(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} \phi(y)\mathrm{d}y = 1 - \overline{\Phi}(x)$. Conditional on $a \leq X \leq b$, X has density

$$\frac{\phi(x)}{\Phi(b)-\Phi(a)} \qquad \text{for } a < x < b \tag{8.1}$$

We denote this truncated normal distribution by $\mathsf{TN}_{a,b}(0, 1)$.

It is well known that if $U \sim U(0, 1)$, the uniform distribution over the interval $(0, 1)$, then

$$X = \Phi^{-1}(\Phi(a) + (\Phi(b) - \Phi(a))U) \tag{8.2}$$

has exactly the standard normal distribution conditional on $a \leq X \leq b$. But even though very accurate approximations are available for Φ and Φ^{-1}, (8.2) is sometimes useless for simulating X. One reason for this is that whenever computations are made under the IEEE-754 double precision standard (which is typical), any number of the form $1 - \epsilon$ for $0 \leq \epsilon < 2 \times 10^{-16}$ (approximately) is identified with 1.0, any positive number smaller than about 10^{-324} cannot be represented at all (it is identified with 0), and numbers smaller than 10^{-308} are represented with less than 52 bits of accuracy.

This implies that $\overline{\Phi}(x) = \Phi(-x)$ is identified as 0 whenever $x \geq 39$ and is identified as 1 whenever $-x \geq 8.3$. Thus, (8.2) cannot work when $a \geq 8.3$. In the latter case, or whenever $a > 0$, it is much better to use the equivalent form:

$$X = -\Phi^{-1}(\overline{\Phi}(a) - (\overline{\Phi}(a) - \overline{\Phi}(b))U), \tag{8.3}$$

which is accurate for a up to about 37, assuming that we use accurate approximations of $\overline{\Phi}(x)$ for $x > 0$ and of $\Phi^{-1}(u)$ for $u < 1/2$. Such accurate approximations are available, for example, in [2] for $\Phi^{-1}(u)$ and via the error function `erf` on most computer systems for $\overline{\Phi}(x)$. For larger values of a (and x), a different inversion approach must be developed, as shown next.

8.2.1 *Inversion Far in the Right Tail*

When $\overline{\Phi}(x)$ is too small to be represented as a floating-point `double`, we will work instead with the Mills' [21] ratio, defined as $q(x) \overset{\text{def}}{=} \overline{\Phi}(x)/\phi(x)$, which is the inverse of the hazard rate (or failure rate) evaluated at x. When x is large, this ratio can be approximated by the truncated series (see [1]):

$$q(x) \approx \frac{1}{x} + \sum_{n=1}^{r} \frac{1 \times 3 \times 5 \times \cdots \times (2n-1)}{(-1)^n x^{2n+1}}. \tag{8.4}$$

In our experiments with $x \geq 10$, we compared $r = 5, 6, 7, 8$, and we found no significant difference (up to machine precision) in the approximation of X in (8.3) by the method we now describe. In view of (8.3), we want to find x such that $\overline{\Phi}(x) = \Phi(-x) = \overline{\Phi}(a) - (\overline{\Phi}(a) - \overline{\Phi}(b))u$, for $0 \leq u \leq 1$, when a is large. This equation can be rewritten as $h(x) = 0$, where

$$h(x) \stackrel{\text{def}}{=} \overline{\Phi}(a) - \overline{\Phi}(x) + (\overline{\Phi}(b) - \overline{\Phi}(a))u \tag{8.5}$$

To solve $h(x) = 0$, we start by finding an approximate solution and then refine this approximation via Newton iterations. We detail how this is achieved. To find an approximate solution, we replace the normal cdf Φ in (8.3) by the standard Rayleigh distribution, whose complementary cdf and density are given by $\overline{F}(x) = \exp(-x^2/2)$ and $f(x) = x\exp(-x^2/2)$ for $x > 0$. Its inverse cdf can be written explicitly as $F^{-1}(u) = (-2\ln(1-u))^{1/2}$. This choice of approximation of Φ^{-1} in the tail has been used before (see, for example, [2] and Sect. 8.4). It is motivated by the facts that $F^{-1}(u)$ is easy to compute and that $\bar{\Phi}(x)/\bar{F}(x) \to 1$ rapidly when $x \to \infty$. By plugging $\overline{F}$ and F^{-1} in place of $\overline{\Phi}$ and Φ^{-1} in (8.3), and solving for x, we find the approximate root

$$x \approx \sqrt{a^2 - 2\ln\left(1 - u + u\exp\left((a^2 - b^2)/2\right)\right)}, \tag{8.6}$$

which is simply the u-th quantile of the standard Rayleigh distribution truncated over (a, b), with density

$$f(x) = \frac{x\exp(-(x^2 - a^2)/2)}{1 - \exp(-(b^2 - a^2)/2)} \qquad \text{for } a < x < b. \tag{8.7}$$

The next step is to improve the approximation (8.6) by applying Newton's method to (8.5). For this, it is convenient to make the change of variable $x = \xi(z)$, where $\xi(z) \stackrel{\text{def}}{=} \sqrt{a^2 - 2\ln(z)}$ and $z = \xi^{-1}(x) = \exp((a^2 - x^2)/2)$, and apply Newton's method to $g(z) \stackrel{\text{def}}{=} h(\xi(z))$. Newton's iteration for solving $g(z) = 0$ has the form $z_{\text{new}} = z - g(z)/g'(z)$, where

$$\begin{aligned}
\frac{g(z)}{g'(z)} &= \frac{h(\xi(z))}{h'(\xi(z))} \cdot \frac{1}{\xi'(z)}, \quad \text{(by the chain rule)} \\
&= z\xi(z)\frac{\overline{\Phi}(\xi(z)) - \overline{\Phi}(a) + u(\overline{\Phi}(a) - \overline{\Phi}(b))}{\phi(\xi(z))} \\
&= x\Big(zq(x) - q(a)(1-u) - q(b)u\exp\left(\tfrac{a^2-b^2}{2}\right)\Big),
\end{aligned}$$

and the identity $x = \xi(z)$ was used for the last equality. A key observation here is that, thanks to the replacement of $\overline{\Phi}$ by q, the computation of $g(z)/g'(z)$ does not involve extremely small quantities that can cause numerical underflow, even for extremely large a.

The complete procedure is summarized in Algorithm 8.1, which we have implemented in Java, MATLAB®, and **R**. According to our experiments, the larger the a, the faster the convergence. For example, for $a = 50$ one requires at most 13 iterations to ensure $\delta_x \le \delta^* = 10^{-10}$, where δ_x represents the relative change in x in the last Newton iteration.

Algorithm 8.1 : Computation of the u-quantile of $\mathsf{TN}_{a,b}(0,1)$

```
Require: Input $u \in (0,1)$, $\delta^*$
  $q_a \leftarrow q(a)$
  $q_b \leftarrow q(b)$
  $c \leftarrow q_a(1-u) + q_b u \exp(\frac{a^2-b^2}{2})$
  $\delta_x \leftarrow \infty$
  $z \leftarrow 1 - u + u \exp(\frac{a^2-b^2}{2})$
  $x \leftarrow \sqrt{a^2 - 2\ln(z)}$
  repeat
    $z \leftarrow z - x(zq(x) - c)$
    $x_{\text{new}} \leftarrow \sqrt{a^2 - 2\ln(z)}$
    $\delta_x \leftarrow |x_{\text{new}} - x|/x$
    $x \leftarrow x_{\text{new}}$
  until $\delta_x \le \delta^*$
  return Quantile $x$
```

We note that for an interval $[a,b] = [a, a+w]$ of fixed length w, when a increases the conditional density concentrates closer to a. In fact, there is practically no difference between generating X conditional on $a \le X \le a+1$ and conditional on $X \ge a$ when $a \ge 30$, but there can be a significant difference for small a.

8.2.2 *Rejection Methods*

We now examine *rejection* (or *acceptance-rejection*) methods, which can be faster than inversion. A large collection of rejection-based generation methods for the normal distribution have been proposed over the years; see [7, 8, 11, 24] for surveys, discussions, comparisons, and tests. Most of them (the fastest ones) use a change of variable and/or precomputed tables to speed up the computations. In its most elementary form, a rejection method to generate from some density f uses a hat function $h \ge f$ and rescales h vertically to a probability density $g = h/\int_{-\infty}^{\infty} h(y)\mathrm{d}y$, often called the proposal density. A random variate X is generated from g, is accepted with probability $f(X)/h(X)$, is rejected otherwise, and the procedure is repeated until X is accepted as the retained realization. In practice, more elaborate versions are used that incorporate transformations and partitions of the area under h.

Any of these proposed rejection methods can be applied easily if $\Phi(b) - \Phi(a)$ is large enough, just by adding a rejection step to reject any value that falls outside $[a, b]$. The acceptance probability for this step is $\Phi(b)-\Phi(a)$. When this probability is too small, this becomes too inefficient and something else must be done. One way is to define a proposal g whose support is exactly $[a, b]$, but this could be inefficient (too much overhead) when a and b change very often. Chopin [7] developed a rejection method specially designed for this situation. The method works by juxtaposing a large number of rectangles of different heights (with equal surface) over some finite interval $[a_{\min}, a_{\max}]$, similar to the trapezoidal approximation in numerical quadrature. However, even Chopin's method achieves efficiency only when it uses an exponential proposal with rate $a = a_{\max}$, when $a_{\max}$ is large enough. Generally, Chopin's trapezoidal method is very fast, and possibly the best method, when $[a_{\min}, a_{\max}]$ contains or is close to zero, but it requires the storage of large precomputed tables. This can be slow on hardware for which memory is limited, like GPUs.

It uses an exponential proposal with rate $a = a_{\max}$ (the RejectTail variant of Algorithms 8.2 below) for the tail above $a_{\max}$ or when $a > a'_{\max}$. The fastest implementation uses 4000 rectangles, $a_{\max} \approx 3.486$, $a'_{\max} \approx 2.605$. This method is fast, although it requires the storage of very large precomputed tables, which could actually slow down computations on certain type of hardware for which memory is limited, like GPUs.

Simple rejection methods for the standard normal truncated to $[a, \infty)$, for $a \geq 0$, have been proposed long ago. Marsaglia [19] proposed a method that uses for g the standard Rayleigh distribution truncated over $[a, \infty)$. An efficient implementation is given in [8, page 381]. Devroye [8, page 382] also gives an algorithm that uses for g an exponential density of rate a shifted by a. These two methods have exactly the same acceptance probability,

$$\alpha(a) = a\sqrt{2\pi}\exp(a^2/2)\overline{\Phi}(a), \tag{8.8}$$

which converges to 1 when $a \to \infty$. Geweke [9] and Robert [22] optimized the acceptance probability to

$$\beta(a) = \lambda\sqrt{2\pi}\exp\left(a\lambda - \lambda^2/2\right)\,\overline{\Phi}(a)$$

by taking the rate $\lambda = (a + \sqrt{a^2+4})/2 > a$ for the shifted exponential proposal. However, the gain with respect to Devroye's method is small and can be wiped out easily by a larger computing time per step. For large a, both are very close to 1 and there is not much difference between them.

We will compare two ways of adapting these methods to a truncation over a finite interval $[a, b]$. The first one is to keep the same proposal g which is positive over the interval $[a, \infty)$ and reject any value generated above b. The second one truncates and rescales the proposal to $[a, b]$ and applies rejection with the truncated proposal. We label them by *RejectTail* and *TruncTail*, respectively. TruncTail has a smaller rejection probability, by the factor $1 - \overline{\Phi}(a)/\overline{\Phi}(b)$, but also entails additional overhead to properly truncate the proposal. Typically, it is worthwhile only if this additional overhead is small and/or the interval $[a, b]$ is very narrow, so it improves the rejection probability significantly. Our experiments will confirm this.

Algorithms 8.2, 8.3, 8.4 state the rejection methods for the TruncTail case with the exponential proposal with rate a [8], with the rate λ proposed in [22], and with the standard Rayleigh distribution, respectively, extended to the case of a finite interval $[a, b]$. For the RejectTail variant, one would remove the computation of q, replace $\ln(1 - qU)$ by $\ln U$, and add $X \le b$ to the acceptance condition. Algorithm 8.5 gives this variant for the Rayleigh proposal.

Algorithm 8.2 : $X \sim \mathsf{TN}_{a,b}(0, 1)$ with exponential proposal with rate a, truncated

```
$K_a \leftarrow 2a^2$
$q \leftarrow 1 - \exp(-(b - a)a)$
**repeat**
    Generate $U, V \sim \mathsf{U}(0, 1)$, independent
    $X \leftarrow -\ln(1 - qU)$
    $E \leftarrow -\ln(V)$
**until** $X^2 \le K_a V$
**return** $a + X/a$
```

Algorithm 8.3 : $X \sim \mathsf{TN}_{a,b}(0, 1)$ with exponential proposal with rate λ, truncated

```
$\lambda \leftarrow (a + \sqrt{a^2 + 4})/2$
$q \leftarrow 1 - \exp(-(b - a)\lambda)$
**repeat**
    Generate $U, V \sim \mathsf{U}(0, 1)$, independent
    $X \leftarrow a - \ln(1 - qU)/\lambda$
**until** $V \le \exp((X - \lambda)^2/2)$
**return** $a + X/a$
```

Algorithm 8.4 : $X \sim \mathsf{TN}_{a,b}(0, 1)$ with Rayleigh proposal, truncated

```
$c \leftarrow a^2/2$
$q \leftarrow 1 - \exp(c - b^2/2)$
**repeat**
    Simulate $U, V \sim \mathsf{U}(0, 1)$, independently.
    $X \leftarrow c - \ln(1 - qU)$
**until** $V^2 X \le a$
**return** $X \leftarrow \sqrt{2X}$
```

Algorithm 8.5 : $X \sim \mathsf{TN}_{a,\infty}(0, 1)$ with Rayleigh proposal and RejectTail

```
c ← $a^2/2$
repeat
    Simulate $U, V \sim \mathsf{U}(0, 1)$, independently.
    $X \leftarrow c - \ln(U)$
until $V^2 X \le a$ and $2X \le b * b$
return $\sqrt{2X}$
```

When the interval $[a, b]$ is very narrow, it makes sense to just use the uniform distribution over this interval for the proposal g. This is suggested in [22] and shown in Algorithm 8.6. Generating from the proposal is then very fast. On the other hand, the acceptance probability may become very small if the interval is far in the tail and $b-a$ is not extremely small. Indeed, the acceptance probability of Algorithm 8.6 is:

$$\frac{\sqrt{2\pi}\,\exp(a^2/2)(\overline{\Phi}(a)-\overline{\Phi}(b))}{b-a} = \frac{q(a)-q(b)\exp((a^2-b^2)/2)}{b-a},$$

which decays at a rate of $1/a$ when $a \to \infty$ while $(b-a) = \mathcal{O}(1)$ remains asymptotically constant ($f(x) = \mathcal{O}(g(x))$ stands for $\lim_{x\uparrow\infty} |f(x)/g(x)| < \infty$). However, when the length of the interval $(b-a) = \mathcal{O}(1/a)$, then the acceptance probability is easily shown to be asymptotically $\mathcal{O}(1)$, rendering Algorithm 8.6 very useful in the tail. In fact, later in Table 8.2 we report that over the interval $[a, b) = [100.0, 100.0001]$ Algorithm 8.6 is decidedly the fastest method.

Algorithm 8.6 : $X \sim \mathsf{TN}_{a,b}(0, 1)$ with uniform proposal, truncated

```
repeat
    Simulate $U, V \sim \mathsf{U}(0, 1)$, independently.
    $X \leftarrow a + (b - a)U$
until $2 \ln V \le a^2 - X^2$
return $X$
```

Another choice that the user can have with those generators (and for any variate generator that depends on some distribution parameters) is to either *precompute* various constants that depend on the parameters and store them in some "distribution" object with fixed parameter values, or to *recompute* these parameter-dependent constants each time a new variate is generated. This type of alternative is common in modern variate generation software [16, 17]. The first approach is worthwhile if the time to compute the relevant constants is significant and several random variates are to be generated with exactly the same distribution parameters. For the applications in Bayesian statistics mentioned earlier, it is typical that the parameters a and b change each time a new variate is generated [7]. But there can be applications in which a large number of variates are generated with the same a and b.

For one-sided intervals $[a, \infty)$, the algorithms can be simplified. One can use the RejectTail framework and since $b = \infty$, there is no need to check if $X \leq b$. When reporting our test results, we label this the *OneSide* case.

Note that computing an exponential is typically more costly than computing a log (by a factor of 2 or 3 for negative exponents and 10 for large exponents, in our experiments) and the latter is more costly than computing a square root (also by a factor of 10). This means significant speedups could be obtained by avoiding the recomputing of the exponential each time at the beginning of Algorithms 8.2, 8.3, and 8.4. This is possible if the same parameter b is used several times, or if $b = \infty$, or if we use RejectTail instead of TruncTail.

8.2.3 Speed Comparisons

We report a representative subset of results of speed tests made with the different methods, for some pairs (a, b). In each case, we generated 10^8 (100 million) truncated normal variates, added them up, printed the CPU time required to do that, and printed the sum for verification. The experiments were made in Java using the SSJ library [16], under Eclipse and Windows 10, on a Lenovo X1 Carbon Thinkpad with an Intel Core(TM) i7-5600U (single) processor running at 2.60 GHz. All programs were executed in a single thread and the CPU times were measured using the stopwatch facilities in class `Chrono` of SSJ, which relies on the `getThreadCpuTime` method from the Java class `ThreadMXBean` to obtain the CPU time consumed so far by a single thread, and subtracts to obtain the CPU time consumed between any two instructions.

The measurements were repeated a few times to verify consistency and varied by about 1–2% at most. The compile times are negligible relative to the reported times. Of course, these timings depend on CPU and memory usage by other processes on the computer, and they are likely to change if we move to a different platform, but on standard processors the relative timings should remain roughly the same. They provide a good idea of what is most efficient to do.

Tables 8.1 and 8.2 report the timings, in seconds. The two columns “recompute” and “precompute” are for the cases where the constants that depend on a and b are recomputed each time a random variate is generated or are precomputed once and for all, respectively, as discussed earlier.

ExponD, ExponR, and Rayleigh refer to the TruncTail versions of Algorithms 8.2, 8.3, and 8.4, respectively. We add “RejectTail” to the name for the RejectTail versions. For ExponRRejectTailLog, we took the log on both sides of the inequality to remove the exponential in the “until” condition. Uniform refers to Algorithm 8.6.

InversionSSJ refers to the default inversion method implemented in SSJ, which uses [2] and gives at least 15 decimal digits of relative precision, combined with a generic (two-sided) “truncated distribution” class also offered in SSJ. InverseQuickSSJ is a faster but much less accurate version based on a cruder

Table 8.1 Time to generate $n = 10^8$ variates for $[a, b] = [3.0, 3.1]$ (left pane) and $[a, b] = [7.0, 8.0]$ (right pane)

Method	CPU time (s)		Method	CPU time	
	recom.	precom.		recom.	precom.
Generation in $[a, b)$			*Generation in* $[a, b)$		
ExponD	6.46	6.22	ExponD	11.70	6.16
ExponDRejectTail	23.04	23.20	ExponDRejectTail	6.04	6.08
ExponR	16.63	9.92	ExponR	15.96	8.98
ExponRRejectTail	32.40	32.40	ExponRRejectTail	9.20	9.09
ExponRRejectTailLog	25.10	25.30	ExponRRejectTailLog	7.03	7.02
Rayleigh	10.29	4.60	Rayleigh	9.86	4.27
RayleighRejectTail	15.23	15.33	RayleighRejectTail	3.91	3.99
Uniform	4.26	4.34	Uniform	25.40	25.68
InverseSSJ	15.14	8.14	InverseSSJ	30.67	8.14
InverseQuickSSJ	18.80	3.31	InverseQuickSSJ	n/a	n/a
InverseRightTail	31.12	7.66	InverseRightTail	31.12	7.70
Generation in $[a, \infty)$			*Generation in* $[a, \infty)$		
ExponDOneSide	6.43	6.46	ExponDOneSide	5.90	5.96
ExponROneSideLog	7.05	6.99	ExponROneSideLog	6.80	6.71
RayleighOneSide	4.07	4.41	RayleighOneSide	3.74	4.05
InverseSSJOneSide	18.81	8.20	InverseSSJOneSide	19.00	8.19
InverseRightTailOneSide	18.72	7.64	InverseRightTailOneSide	18.76	7.59

Table 8.2 Time to generate $n = 10^8$ variates for $[a, b] = [100.0, 102.0]$ (left pane) and $[a, b] = [100.0, 100.0001]$ (right pane)

Method	CPU time (s)		Method	CPU time	
	recom.	precom.		recom.	precom.
Generation in $[a, b)$			*Generation in* $[a, b)$		
ExponD	11.68	6.01	ExponD	12.31	6.83
ExponDRejectTail	5.88	5.91	ExponDRejectTail	543.80	546.58
ExponR	15.79	8.86	ExponR	16.47	10.65
ExponRRejectTail	9.13	9.02	ExponRRejectTail	865.24	865.34
ExponRRejectTailLog	6.93	6.96	ExponRRejectTailLog	651.19	648.99
Rayleigh	9.97	4.16	Rayleigh	10.59	5.07
RayleighRejectTail	3.84	3.90	RayleighRejectTail	323.08	322.41
Uniform	650.62	656.42	Uniform	3.59	3.62
InverseMillsRatio	22.31	15.97	InverseMillsRatio	18.03	12.12
Generation in $[a, \infty)$			*Generation in* $[a, \infty)$		
ExponDOneSide	5.77	5.82	ExponDOneSide	5.79	5.83
ExponROneSideLog	6.72	6.63	ExponROneSideLog	6.74	6.63
RayleighOneSide	3.67	3.96	RayleighOneSide	3.66	3.99
InverseMillsRatioOneSide	15.62	15.84	InverseMillsRatioOneSide	15.67	15.84

approximation of $\overline{\Phi}$ from [20] based on table lookups, which returns about six decimal digits of precision. We do not recommend it, due to its low accuracy. Moreover, the implementation we used does not handle well values larger than about 5 in the right tail, so we report results only for small a. InverseRightTail uses the accurate approximation of $\overline{\Phi}$ together with (8.3). InverseMillsRatio is our new inversion method based on Mills ratio, with $\delta^* = 10^{-10}$. This method is designed for the case where a is large, and our implementation is designed to be accurate for $a \geq 10$, so we do not report results for it in Table 8.1. For all the methods, we add "OneSide" for the simplified OneSide versions, for which $b = \infty$.

For the OneSide case, that is, $b = \infty$, the Rayleigh proposal gives the fastest method in all cases, and there is no significant gain in precomputing and storing the constant $c = a^2/2$.

For finite intervals $[a, b]$, when $b - a$ is very small so $\overline{\Phi}(b)/\overline{\Phi}(a)$ is close to 1, the uniform proposal wins and the RejectTail variants are very slow. See right pane of Table 8.2. Precomputing the constants is also not useful for the uniform proposal. For larger intervals in the tail, $\overline{\Phi}(x)$ decreases quickly at the beginning of the interval and this leads to very low acceptance ratios; see right pane of Table 8.1 and left pane of Table 8.2. A Rayleigh proposal with the RejectTail option is usually the fastest method in this case. Precomputing and storing the constants is also not very useful for this option. For intervals closer to the center, as in the left pane of Table 8.1, the uniform proposal performs well for larger (but not too large) intervals, and the RejectTail option becomes slower unless $[a, b]$ is very wide. The reason is that for a fixed $w > 0$, $\overline{\Phi}(a + w)/\overline{\Phi}(a)$ is larger (closer to 1) when $a > 0$ is closer to 0.

8.3 Simulation from the Tail of the Multivariate Normal

Let $\phi_\Sigma(\boldsymbol{y})$ and

$$\overline{\Phi}_\Sigma(\boldsymbol{a}) = \mathbb{P}[\boldsymbol{Y} \geq \boldsymbol{a}], \quad \boldsymbol{Y} \sim \mathsf{N}(\boldsymbol{0}, \Sigma),$$

denote the density and tail distribution, respectively, of the multivariate $\mathsf{N}(\boldsymbol{0}, \Sigma)$ distribution with (positive-definite) covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. In the multivariate extension to (8.1), we wish to simulate from the pdf ($\mathbb{I}\{\cdot\}$ is the indicator function):

$$\frac{\phi_\Sigma(\boldsymbol{y})\mathbb{I}\{\boldsymbol{y} \geq \boldsymbol{a}(\gamma)\}}{\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))}, \tag{8.9}$$

where $\max_i a_i > 0$, and γ is a *tail parameter* such that at least one component of $\boldsymbol{a}(\gamma)$ diverges to ∞ when $\gamma \to \infty$ (that is, $\lim_{\gamma \uparrow \infty} \|\boldsymbol{a}(\gamma)\| = \infty$, see [10]). To simulate from this conditional density, we describe a rejection algorithm that uses an optimally designed multivariate exponential proposal. Interestingly, unlike the truncated exponential proposal in the one-dimensional setting (see Algorithms 8.2

and 8.3), our multivariate exponential proposal is not truncated. Before giving the details of the rejection algorithm, we need to introduce some preliminary theory and notation.

8.3.1 Preliminaries and Notation

Define P as a permutation matrix, which maps $(1, \ldots, d)^\top$ into the permutation vector $\boldsymbol{p} = (p_1, \ldots, p_d)^\top$, that is, $\mathrm{P}(1, \ldots, d)^\top = \boldsymbol{p}$. Then, $\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma)) = \mathbb{P}(\mathrm{P}\boldsymbol{Y} \geq \mathrm{P}\boldsymbol{a}(\gamma))$ and $\mathrm{P}\boldsymbol{Y} \sim \mathsf{N}(\boldsymbol{0}, \mathrm{P}\Sigma\mathrm{P}^\top)$ for any $\boldsymbol{p}$. We will specify $\boldsymbol{p}$ shortly.

First, define the constrained (convex) quadratic optimization:

$$\begin{aligned} &\min_{\boldsymbol{y}} \; \frac{1}{2}\boldsymbol{y}^\top \left(\mathrm{P}\Sigma\mathrm{P}^\top\right)^{-1} \boldsymbol{y} \\ &\text{subject to: } \boldsymbol{y} \geq \mathrm{P}\boldsymbol{a}(\gamma). \end{aligned} \tag{8.10}$$

Suppose $\boldsymbol{\lambda} \in \mathbb{R}^d$ is the Lagrange multiplier vector, associated with (8.10). Partition the vector as $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_1^\top, \boldsymbol{\lambda}_2^\top)^\top$ with $\dim(\boldsymbol{\lambda}_1) = d_1$ and $\dim(\boldsymbol{\lambda}_2) = d_2$, where $d_1 + d_2 = d$. In the same way, partition vectors $\boldsymbol{y}$, $\boldsymbol{a}$, and matrix

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}. \tag{8.11}$$

We now observe that we can select the permutation vector $\boldsymbol{p}$ and the corresponding matrix P so that all the d_1 active constraints in (8.10) correspond to $\boldsymbol{\lambda}_1 > \boldsymbol{0}$ and all the d_2 inactive constraints correspond to $\boldsymbol{\lambda}_2 = \boldsymbol{0}$. Without loss of generality, we can thus assume that $\boldsymbol{a}$ and Σ are reordered via the permutation matrix P as a preprocessing step. After this preprocessing step, the solution $\boldsymbol{y}^*$ of (8.10) with $\mathrm{P} = \mathrm{I}$ will satisfy $\boldsymbol{y}_1^* = \boldsymbol{a}_1$ (active constraints: $\boldsymbol{\lambda}_1 > \boldsymbol{0}$) and $\boldsymbol{y}_2^* > \boldsymbol{a}_2$ (inactive constraints: $\boldsymbol{\lambda}_2 = \boldsymbol{0}$).

We also assume that for large enough γ, the active constraint set of (8.10) becomes independent of γ, see [10]. An example is given in Corollary 8.1 below.

8.3.2 The Rejection Algorithm

First, we note that simulating $\boldsymbol{Y}$ from (8.9) is equivalent to simulating $\boldsymbol{X} \sim \mathsf{N}(-\boldsymbol{a}(\gamma), \Sigma)$, conditional on $\boldsymbol{X} \geq \boldsymbol{0}$, and then delivering $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{a}$. Thus, our initial goal is to simulate from the target:

$$\pi(\boldsymbol{x}) = \frac{\phi_\Sigma(\boldsymbol{x} + \boldsymbol{a}(\gamma))\mathbb{I}\{\boldsymbol{x} \geq \boldsymbol{0}\}}{\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))}.$$

Second, the partitioning into active and inactive constraints of (8.10) suggests the following proposal density: $g(\boldsymbol{x};\boldsymbol{\eta}) = g_1(\boldsymbol{x}_1;\boldsymbol{\eta})g_2(\boldsymbol{x}_2|\boldsymbol{x}_1),\ \boldsymbol{\eta} > \boldsymbol{0}$, where

$$g_1(\boldsymbol{x}_1;\boldsymbol{\eta}) = \exp(-\boldsymbol{\eta}^\top \boldsymbol{x}_1)\prod_{k=1}^{d_1}\eta_k, \quad \boldsymbol{x}_1 \geq \boldsymbol{0}$$

is a multivariate exponential proposal, and

$$g_2(\boldsymbol{x}_2|\boldsymbol{x}_1) = \frac{\phi_\Sigma(\boldsymbol{x}+\boldsymbol{a})}{\phi_{\Sigma_{11}}(\boldsymbol{x}_1+\boldsymbol{a}_1)}$$

is the multivariate normal pdf of $\boldsymbol{x}_2$, conditional on $\boldsymbol{x}_1$ (see [12, Page 146]):

$$\boldsymbol{X}_2|(\boldsymbol{X}_1 = \boldsymbol{x}_1) \sim \mathsf{N}(-\boldsymbol{a}_2 + \Sigma_{12}^\top\Sigma_{11}^{-1}(\boldsymbol{x}_1+\boldsymbol{a}_1),\ \Sigma_{22} - \Sigma_{12}^\top\Sigma_{11}^{-1}\Sigma_{12}).$$

With this proposal, the likelihood ratio for acceptance–rejection is

$$\frac{\pi(\boldsymbol{x})\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))}{g(\boldsymbol{x};\boldsymbol{\eta})} = \mathbb{I}\{\boldsymbol{x}\geq\boldsymbol{0}\}\frac{\phi_{\Sigma_{11}}(\boldsymbol{x}_1+\boldsymbol{a}_1)}{g_1(\boldsymbol{x}_1;\boldsymbol{\eta})} = \mathbb{I}\{\boldsymbol{x}\geq\boldsymbol{0}\}\exp\left(\psi(\boldsymbol{x}_1;\boldsymbol{\eta})\right),$$

where ψ is defined as

$$\psi(\boldsymbol{x}_1;\boldsymbol{\eta}) := -\frac{(\boldsymbol{x}_1+\boldsymbol{a}_1)^\top\Sigma_{11}^{-1}(\boldsymbol{x}_1+\boldsymbol{a}_1)}{2} + \boldsymbol{\eta}^\top\boldsymbol{x}_1 - \sum_{k=1}^{d_1}\ln(\eta_k)$$
$$-\frac{\ln|\Sigma_{11}|}{2} - \frac{d_1\ln(2\pi)}{2}.$$

Next, our goal is to select the value for $\boldsymbol{\eta}$ that will maximize the acceptance rate of the resulting rejection algorithm (see Algorithm 8.7 below).

It is straightforward to show that, with the given proposal density, the acceptance rate for a fixed $\boldsymbol{\eta} > \boldsymbol{0}$ is given by

$$\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))\exp(-\max\nolimits_{\boldsymbol{x}_1\geq\boldsymbol{0}}\psi(\boldsymbol{x}_1;\boldsymbol{\eta})).$$

Hence, to maximize the acceptance rate, we minimize $\max_{\boldsymbol{x}_1\geq\boldsymbol{0}}\psi(\boldsymbol{x}_1;\boldsymbol{\eta})$ with respect to $\boldsymbol{\eta}$. In order to compute the minimizing $\boldsymbol{\eta}$, we exploit a few of the properties of ψ.

The most important property is that ψ is concave in $\boldsymbol{x}_1$ for every $\boldsymbol{\eta}$, and that ψ is convex in $\boldsymbol{\eta}$ for every $\boldsymbol{x}_1$. Moreover, ψ is continuously differentiable in $\boldsymbol{\eta}$, and we have the saddle-point property (see [3]):

$$\min_{\boldsymbol{\eta}>\boldsymbol{0}}\max_{\boldsymbol{x}_1\geq\boldsymbol{0}}\psi(\boldsymbol{x}_1;\boldsymbol{\eta}) = \max_{\boldsymbol{x}_1\geq\boldsymbol{0}}\min_{\boldsymbol{\eta}>\boldsymbol{0}}\psi(\boldsymbol{x}_1;\boldsymbol{\eta}). \tag{8.12}$$

Let $\psi^* = \psi(\boldsymbol{x}_1^*; \boldsymbol{\eta}^*)$ denote the optimum of the minimax optimization (8.12) at the solution $\boldsymbol{x}_1^*$ and $\boldsymbol{\eta}^*$. The right-hand-side of (8.12) suggests a method for computing $\boldsymbol{\eta}^*$, namely, we can first minimize with respect to $\boldsymbol{\eta}$ (this gives $\boldsymbol{\eta} = \mathbf{1}/\boldsymbol{x}_1$, where the vector division is componentwise), and then maximize over $\boldsymbol{x}_1 \geq \mathbf{0}$. This yields the concave (unconstrained) optimization program for $\boldsymbol{x}_1^*$:

$$\boldsymbol{x}_1^* = \operatorname{argmax}\left\{-\frac{(\boldsymbol{x}_1+\boldsymbol{a}_1)^\top \Sigma_{11}^{-1}(\boldsymbol{x}_1+\boldsymbol{a}_1)}{2} + \sum_{k=1}^{d_1} \ln x_k\right\}. \tag{8.13}$$

It then follows that $\boldsymbol{\eta}^* = \mathbf{1}/\boldsymbol{x}_1^*$. In summary, we have the following algorithm for simulation from (8.9).

Algorithm 8.7 : $\boldsymbol{X} \sim \mathsf{N}(\mathbf{0}, \Sigma)$ conditional on $\boldsymbol{X} \geq \boldsymbol{a}(\gamma)$, for large γ

```
Solve (8.10) with P = I and compute the associated Lagrange multiplier $\boldsymbol{\lambda}$. Using $\boldsymbol{\lambda}$, construct
the reordering (permutation) matrix P, if needed.
$\boldsymbol{a} \leftarrow \mathrm{P}\boldsymbol{a}$
$\Sigma \leftarrow \mathrm{P}\Sigma\mathrm{P}^\top$
Let L be the lower triangular Cholesky factor of $\Sigma_{22} - \Sigma_{12}^\top\Sigma_{11}^{-1}\Sigma_{12}$, see (8.11)
Solve the concave optimization problem (8.13) to obtain $\boldsymbol{x}_1^*$
$\boldsymbol{\eta}_1^* \leftarrow \mathbf{1}/\boldsymbol{x}_1^*$
$\psi^* \leftarrow \psi(\boldsymbol{x}_1^*; \boldsymbol{\eta}^*)$
repeat
    repeat
        Simulate $U_0, U_1, \ldots, U_{d_1} \sim \mathsf{U}(0, 1)$, independently
        $E_k \leftarrow -\ln(U_k)/\eta_k^*$ for $k = 1, \ldots, d_1$
        $\boldsymbol{X}_1 \leftarrow (E_1, \ldots, E_{d_1})^\top$        {simulate $\boldsymbol{X}_1 \sim g_1(\boldsymbol{x}_1; \boldsymbol{\eta}^*)$}
        $E \leftarrow -\ln(U_0)$
    until $E > \psi^* - \psi(\boldsymbol{X}_1; \boldsymbol{\eta}^*)$
    $\boldsymbol{Z}_2 \leftarrow (Z_1, \ldots, Z_{d_2})^\top$, where $Z_1, \ldots, Z_{d_2} \sim \mathsf{N}(0, 1)$, independently.
    $\boldsymbol{X}_2 \leftarrow \mathrm{L}\boldsymbol{Z}_2 - \boldsymbol{a}_2 + \Sigma_{12}^\top\Sigma_{11}^{-1}(\boldsymbol{X}_1 + \boldsymbol{a}_1)$       {simulate $\boldsymbol{X}_2 \sim g_2(\boldsymbol{x}_2|\boldsymbol{X}_1)$}
until $\boldsymbol{X}_2 \geq \mathbf{0}$
$\boldsymbol{X} \leftarrow \boldsymbol{X} + \boldsymbol{a}$       {shift to obtain draw from pdf (8.9)}
$\boldsymbol{X} \leftarrow \mathrm{P}^\top\boldsymbol{X}$      {reverse reordering, if any}
return $\boldsymbol{X}$
```

8.3.3 Asymptotic Efficiency

The acceptance rate of Algorithm 8.7 above is

$$\mathbb{P}_g[E > \psi^* - \psi(\boldsymbol{X}_1; \boldsymbol{\eta}); \boldsymbol{X}_2 \geq \mathbf{0}] = \overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))\exp(-\psi^*),$$

where $\mathbb{P}_g$ indicates that $\boldsymbol{X}$ was drawn from the proposal $g(\boldsymbol{x}; \boldsymbol{\eta}^*)$. As in the one-dimensional case, see (8.8), it is of interest to find out how this rate depends on the tail parameter γ. In particular, if the acceptance rate decays to zero rapidly as $\gamma \uparrow \infty$, then Algorithm 8.7 will not be a viable algorithm for simulation from the tail of the multivariate Gaussian. Fortunately, the following result asserts that the acceptance rate does not decay to zero as we move further and further into the tail of the Gaussian.

Theorem 8.1 (Asymptotically Bounded Acceptance Rate) *Let $\boldsymbol{y}^*$ be the solution to (8.10) after any necessary reordering via permutation matrix* P. *Define $\boldsymbol{a}_\infty := \lim_{\gamma\uparrow\infty}(\boldsymbol{a}_2(\gamma) - \boldsymbol{y}_2^*(\gamma))$ with $\boldsymbol{a}_\infty \leq \boldsymbol{0}$. Then, the acceptance rate of the rejection Algorithm 8.7 is ultimately bounded from below:*

$$\liminf_{\gamma\uparrow\infty} \overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma)) \exp(-\psi^*(\gamma)) \geq \mathbb{P}[\boldsymbol{Y}_2 \geq \boldsymbol{a}_\infty \,|\, \boldsymbol{Y}_1 = \boldsymbol{0}],$$

where the probability $\mathbb{P}[\boldsymbol{Y}_2 \geq \boldsymbol{a}_\infty \,|\, \boldsymbol{Y}_1 = \boldsymbol{0}]$ is calculated under the original measure (that is, $\boldsymbol{Y} \sim \mathsf{N}(\boldsymbol{0}, \Sigma)$) and, importantly, does not depend on γ.

Proof First, note that with the assumptions and notation of Sect. 8.3.1, Hashorva and Hüsler [10] have shown the following:

$$\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma)) = \frac{\mathbb{P}[\boldsymbol{Y}_2 \geq \boldsymbol{a}_\infty \,|\, \boldsymbol{Y}_1 = \boldsymbol{0}]}{(2\pi)^{d_1/2}|\Sigma_{11}|^{1/2}\prod_{k=1}^{d_1} \boldsymbol{e}_k^\top \Sigma_{11}^{-1}\boldsymbol{a}_1} \exp\left(-\frac{\boldsymbol{a}_1^\top \Sigma_{11}^{-1}\boldsymbol{a}_1}{2}\right)(1+o(1)), \quad \gamma \uparrow \infty,$$

where $\boldsymbol{e}_k$ is the unit vector with a 1 in the k-th position, and $f(x) = o(g(x))$ stands for $\lim_{x\to a} f(x)/g(x) = 0$.

Second, the saddle-point property (8.12) implies the following sequence of inequalities for any arbitrary $\boldsymbol{\eta}$: $\psi^* \leq \psi(\boldsymbol{x}_1^*; \boldsymbol{\eta}) \leq \max_{\boldsymbol{x}_1} \psi(\boldsymbol{x}_1; \boldsymbol{\eta})$. In particular, when $\boldsymbol{\eta} = \Sigma_{11}^{-1}\boldsymbol{a}_1$, then $\max_{\boldsymbol{x}_1} \psi(\boldsymbol{x}_1; \Sigma_{11}^{-1}\boldsymbol{a}_1) = \psi(\boldsymbol{0}; \Sigma_{11}^{-1}\boldsymbol{a}_1)$, and we obtain:

$$\exp(-\psi^*) \geq \exp(-\psi(\boldsymbol{0}; \Sigma_{11}^{-1}\boldsymbol{a}_1)) = \frac{\prod_{k=1}^{d_1} \boldsymbol{e}_k^\top \Sigma_{11}^{-1}\boldsymbol{a}_1}{\phi_{\Sigma_{11}}(\boldsymbol{a}_1)}$$

Therefore, $\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma)) \exp(-\psi^*) \geq \mathbb{P}[\boldsymbol{Y}_2 \geq \boldsymbol{a}_\infty \,|\, \boldsymbol{Y}_1 = \boldsymbol{0}](1+o(1))$ as $\gamma \uparrow \infty$, and the result of the theorem follows. ■

As a special case, we consider the asymptotic result of Savage [23]:

$$\frac{\overline{\Phi}_\Sigma(\gamma\,\Sigma\boldsymbol{c})}{\phi_\Sigma(\gamma\,\Sigma\boldsymbol{c})} = \frac{1}{\gamma^d \prod_{k=1}^{d} c_k}(1+o(1)), \quad \boldsymbol{c} > \boldsymbol{0}, \quad \gamma \uparrow \infty, \tag{8.14}$$

which is the multivariate extension of the one-dimensional Mills' ratio [21]: $\frac{\overline{\Phi}(\gamma)}{\phi(\gamma)} = \frac{1}{\gamma}(1+o(1))$. Interestingly, the following corollary shows that when the tail is of the Savage-Mills type, the acceptance probability not only remains bounded away from zero, but approaches unity.

Table 8.3 Estimates of the acceptance probability, $\overline{\Phi}_\Sigma(\boldsymbol{a}(\gamma))\exp(-\psi^*)$, as a function of γ

γ	10	15	20	25	30	50	100	10^3
Accept. rate	0.009	0.04	0.0815	0.15	0.19	0.34	0.44	0.50

Corollary 8.1 (Acceptance with Probability One) *The acceptance rate of Algorithm 8.7 for simulation from (8.9) with $\boldsymbol{a} = \gamma\Sigma\boldsymbol{c}$ for some $\boldsymbol{c} > \boldsymbol{0}$ satisfies:*

$$\lim_{\gamma\uparrow\infty} \overline{\Phi}_\Sigma(\gamma\Sigma\boldsymbol{c})\exp(-\psi^*(\gamma)) = 1$$

Proof Straightforward computations show that the Lagrange multiplier of (8.10) (with P = I, the identity matrix) is $\boldsymbol{\lambda} = \Sigma^{-1}\boldsymbol{a} = \gamma\boldsymbol{c} > \boldsymbol{0}$, so that the set of inactive constraints is empty. Then, repeating the argument in Theorem 8.1:

$$\exp(-\psi^*) \geq \exp(-\psi(\boldsymbol{0},\gamma\boldsymbol{c})) = \frac{\gamma^d\prod_{k=1}^d c_k}{\phi_\Sigma(\gamma\Sigma\boldsymbol{c})} \overset{(8.14)}{=} \frac{1+o(1)}{\overline{\Phi}_\Sigma(\gamma\Sigma\boldsymbol{c})},$$

as desired. ■

As a numerical example, we used Algorithm 8.7 to simulate 10^3 random vectors from (8.9) for $d = 10$, $\boldsymbol{a} = \gamma\boldsymbol{1}$, and $\Sigma = \frac{9}{10}\boldsymbol{1}\boldsymbol{1}^\top + \frac{1}{10}\mathrm{I}$ (strong positive correlation) for a range of large values of γ.

Table 8.3 above reports the acceptance rate, estimated by observing the proportion of rejected proposals in line 17 of Algorithm 8.7, for a range of different γ.

The table confirms that as γ gets larger, the acceptance rate improves.

8.4 Conclusion

We have proposed and tested inversion and rejection methods to generate a standard normal, truncated to an interval $[a, b]$, when $a \gg 0$. We also proposed a rejection method for the tail of the multivariate normal distribution.

In the univariate setting, inversion is slower than the fastest rejection method, as expected. However, inversion is still desirable in many situations. Our new inversion method excels in those situations when a is large (say, $a \geq 10$). For a not too large (say, $a \leq 30$), the accurate approximation of [2] implemented in InversionSSJ works well.

When inversion is not needed, the rejection method with the Rayleigh proposal is usually the fastest when a is large enough. especially if a large number of variates must be generated for the same interval $[a, b]$, in which case the cost of precomputing the constants used in the algorithm can be amortized over many calls.

It is interesting to see that in the univariate setting, using the Rayleigh proposal is faster than using the truncated exponential proposal as in [7, 9, 22]. The RejectTail variant is usually the fastest, unless $\bar{\Phi}(b)/\bar{\Phi}(a)$ is far from 0, which happens when the interval $[a, b]$ is very narrow or a is not large (say $a \leq 5$).

However, in the multivariate setting, we show that the *truncated* exponential method of [7, 9, 22] can be extended to help simulate from the multivariate normal tail, provided that we use an *untruncated* multivariate exponential proposal (that is, $\boldsymbol{X} \geq \mathbf{0}$) combined with a shift of the Gaussian mean (that is, $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{a}$).

References

1. M. Abramowitz, I.A. Stegun, *Handbook of Mathematical Functions* (Dover, New York, 1970)
2. J.M. Blair, C.A. Edwards, J.H. Johnson, Rational Chebyshev approximations for the inverse of the error function. Math. Comput. **30**, 827–830 (1976)
3. Z.I. Botev, The normal law under linear restrictions: simulation and estimation via minimax tilting. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **79**(1), 125–148 (2017)
4. Z.I. Botev, P. L'Ecuyer, Efficient estimation and simulation of the truncated multivariate Student-t distribution, in *Proceedings of the 2015 Winter Simulation Conference* (IEEE Press, Piscataway, 2015), pp. 380–391
5. Z.I. Botev, P. L'Ecuyer, Simulation from the normal distribution truncated to an interval in the tail, in *10th EAI International Conference on Performance Evaluation Methodologies and Tools*, 25th–28th October 2016 Taormina (ACM, New York, 2017), pp. 23–29
6. Z.I. Botev, M. Mandjes, A. Ridder, Tail distribution of the maximum of correlated Gaussian random variables, in *Proceedings of the 2015 Winter Simulation Conference* (IEEE Press, Piscataway, 2015), pp. 633–642
7. N. Chopin, Fast simulation of truncated Gaussian distributions. Stat. Comput. **21**(2), 275–288 (2011)
8. L. Devroye, *Non-Uniform Random Variate Generation* (Springer, New York, NY, 1986)
9. J. Geweke, Efficient simulation of the multivariate normal and Student-t distributions subject to linear constraints and the evaluation of constraint probabilities, in *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, Fairfax, VA, 1991, pp. 571–578
10. E. Hashorva, J. Hüsler, On multivariate Gaussian tails. Ann. Inst. Stat. Math. **55**(3), 507–522 (2003)
11. W. Hörmann, J. Leydold, G. Derflinger, *Automatic Nonuniform Random Variate Generation* (Springer, Berlin, 2004)
12. D.P. Kroese, T. Taimre, Z.I. Botev, *Handbook of Monte Carlo Methods* (Wiley, New York, 2011)
13. P. L'Ecuyer, Variance reduction's greatest hits, in *Proceedings of the 2007 European Simulation and Modeling Conference*, Ghent (EUROSIS, Hasselt, 2007), pp. 5–12
14. P. L'Ecuyer, Quasi-Monte Carlo methods with applications in finance. Finance Stochast. **13**(3), 307–349 (2009)
15. P. L'Ecuyer, Random number generation with multiple streams for sequential and parallel computers, in *Proceedings of the 2015 Winter Simulation Conference*, pp. 31–44 (IEEE Press, New York, 2015)
16. P. L'Ecuyer, SSJ: stochastic simulation in Java, software library (2016). http://simul.iro.umontreal.ca/ssj/
17. J. Leydold, UNU.RAN—Universal Non-Uniform RANdom number generators (2009). Available at http://statmath.wu.ac.at/unuran/
18. G. Marsaglia, Generating a variable from the tail of the normal distribution. Technometrics **6**(1), 101–102 (1964)
19. G. Marsaglia, T.A. Bray, A convenient method for generating normal variables. SIAM Rev. **6**, 260–264 (1964)
20. G. Marsaglia, A. Zaman, J.C.W. Marsaglia, Rapid evaluation of the inverse normal distribution function. Stat. Probab. Lett. **19**, 259–266 (1994)

21. J.P. Mills, Table of the ratio: area to bounding ordinate, for any portion of normal curve. Biometrika **18**(3/4), 395–400 (1926)
22. C.P. Robert, Simulation of truncated normal variables. Stat. Comput. **5**(2), 121–125 (1995)
23. R.I. Savage, Mills' ratio for multivariate normal distributions. J. Res. Nat. Bur. Standards Sect. B **66**, 93–96 (1962)
24. D.B. Thomas, W. Luk, P.H. Leong, J.D. Villasenor, Gaussian random number generators. ACM Comput. Surv. **39**(4), Article 11 (2007)