Simulation from the Normal Distribution Truncated to an Interval far in the Tail

Zdravko Botev

University of New South Wales, Sydney, Australia

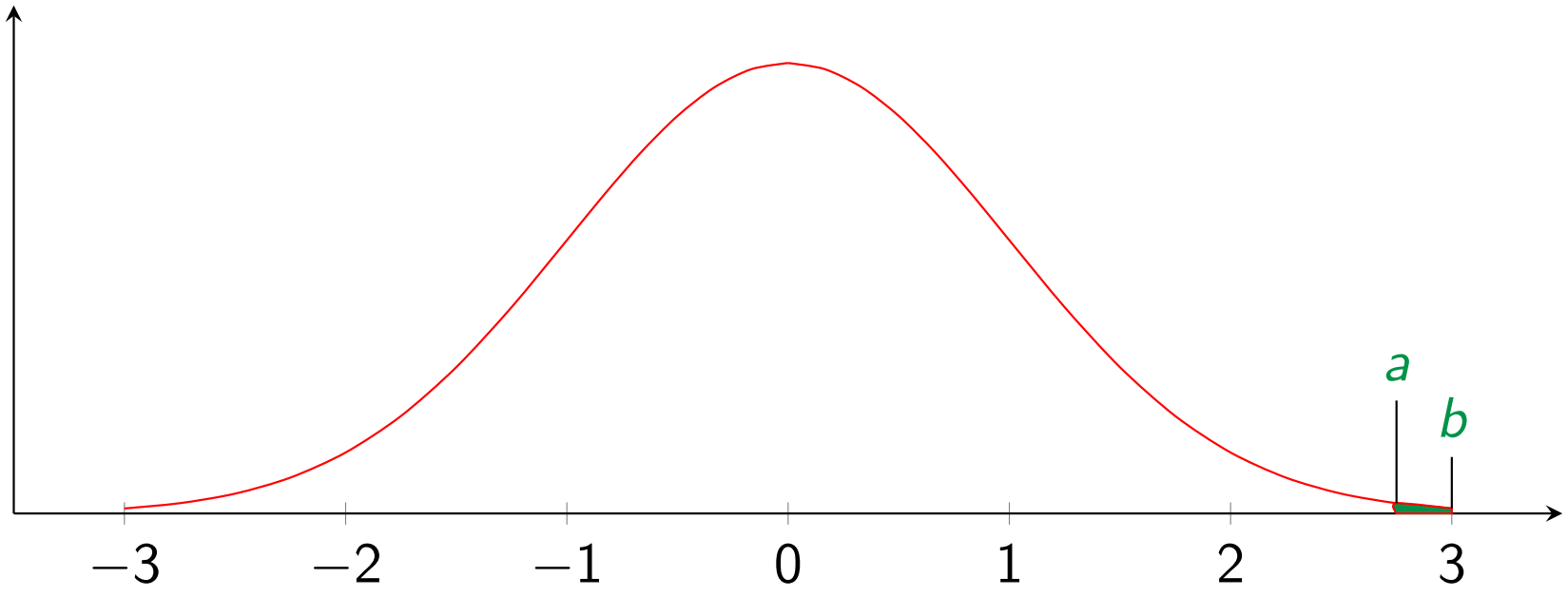
Pierre L'Ecuyer

Valuetools, Taormina, Italia, October 2016

Problem considered

Let $0 \ll a < b$ and we want to generate a random variate X from the **standard normal density truncated to the interval** $[a, b]$**.**

The case where $a < b \ll 0$ can be treated by symmetry.
The case where a or b is near 0 or $a < 0 < b$ is easier and can be handled by standard methods.



We distinguish two situations:

A. In the first, it is required that X be generated by inversion.
For example, inversion is often required in the context of quasi-Monte Carlo, derivative estimation by finite differences, comparisons with common random numbers, etc.

B. In case any accurate method is acceptable, then we can use rejection.

We compare the best available methods for each situation.
We also propose a new inversion method for the far tail.

Why do this?

For applications in Bayesian statistics and computational biology, for example, it is often required to generate $\mathbf{X}$ from the multinormal or Student-t distribution conditional on $\mathbf{a} \leq \mathbf{X} \leq \mathbf{b}$, or conditional on $\mathbf{X}$ satisfying a set of linear inequalities.

Efficient simulation-based methods for this require repeated draws from normal distributions truncated to different intervals $[a, b]$, often far in the tail.

Generating a normal $Y \sim \mathrm{N}(\mu, \sigma^2)$ truncated to $[a', b']$ is equivalent to generating a standard normal X truncated to $[a, b] = [(a' - \mu)/\sigma, (b' - \mu)/\sigma]$ and putting $Y = \sigma X + \mu$.
So it suffices to consider the standard normal.

Basic Inversion and Notation

Let ϕ be the standard normal density, Φ its cdf, $\overline{\Phi} = 1 - \Phi$ the complementary cdf, and Φ^{-1} the inverse cdf. We have

$$\Phi^{-1}(u) = \min\{x \in \mathbb{R} \mid \Phi(x) \geq u\}$$

and if $X \sim \mathrm{N}(0,1)$,

$$\Phi(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} \phi(y)\mathrm{d}y = 1 - \overline{\Phi}(x).$$

Conditional on $a \leq X \leq b$, X has the $\mathrm{TN}_{a,b}(0,1)$ density

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

If $U \sim U(0,1)$, then

$$X = \Phi^{-1}[\Phi(a) + (\Phi(b) - \Phi(a))U] \sim \mathrm{TN}_{a,b}(0,1).$$

Very accurate approximations are available for $\Phi(x)$ and $\Phi^{-1}(x)$, but there are nasty numerical problems when x is large.

Under the IEEE-754 double precision standard, $1 - \epsilon$ is identified with 1 whenever $0 \leq \epsilon < 2 \times 10^{-16}$ (approximately).
For this reason, $\Phi(x)$ is identified with 1 whenever $x > 8.3$ (approx.).
Thus, direct inversion cannot work when $a \geq 8.3$.

For $a > 0$, it is better to use:

$$X = -\Phi^{-1}[\overline{\Phi}(a) - (\overline{\Phi}(a) - \overline{\Phi}(b))U]$$

instead, which can be accurate for a up to about 37 if 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, but not always used.

For larger a, we have a problem because $\overline{\Phi}(a)$ is too small. In the IEEE standard, positive numbers smaller than about 10^{-324} cannot be represented at all (are identified with 0), and numbers smaller than 10^{-308} are represented with less than 52 bits of accuracy. For $x \geq 39$, we have $\overline{\Phi}(a) < 10^{-324}$, so we need a different way to implement inversion.

Inversion far in the Right Tail

For large x, since $\overline{\Phi}(x)$ is too small and nor representable, we will work instead with the Mills ratio $q(x) \stackrel{\text{def}}{=} \overline{\Phi}(x)/\phi(x)$. For large x, one has

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

This series diverges when $r \to \infty$ for fixed x, but it gives a lower bound when r is odd and an upper bound when r is even. The distance between the lowest upper bound and the highest lower bound converges to 0 rapidly when x increases. For $x \geq 10$, taking $r = 6$ is sufficient.

Inversion amounts to finding the root x of

$$h(x) \stackrel{\text{def}}{=} \overline{\Phi}(a) - \overline{\Phi}(x) + (\overline{\Phi}(b) - \overline{\Phi}(a))u = 0.$$

We start with an approximate solution x_0 and refine it by Newton iterations.

To get x_0, we replace $\overline{\Phi}$ in $h(x)$ by the complementary cdf of the standard Rayleigh distribution, $\overline{F}(x) = 1 - F(x) = \exp(-x^2/2)$ for $x \geq 0$. This provides a good approximation for large x because $\bar{\Phi}(x)/\bar{F}(x) \to 1$ rapidly when $x \to \infty$. Solving yields

$$x \approx x_0 = (a^2 - 2\ln\left(1 - u + u\exp\left((a^2 - b^2)/2\right)\right))^{1/2},$$

the u-th quantile of the truncated Rayleigh distribution.

Then we make the change of variable $x = \xi(z) \stackrel{\text{def}}{=} \sqrt{a^2 - 2\ln(z)}$ and apply Newton's method to find the root of $g(z) \stackrel{\text{def}}{=} h(\xi(z)) = 0$ in terms of z:

$$z_{\text{new}} = z - g(z)/g'(z),$$

where

$$\frac{g(z)}{g'(z)} = x\left(zq(x) - q(a)(1-u) - q(b)u\exp\left(\tfrac{a^2-b^2}{2}\right)\right).$$

Computing $g(z)/g'(z)$ involve no very small quantity.

InverseMillsRatio: Returns the u-quantile of $\mathrm{TN}_{a,b}(0,1)$

$q_a \leftarrow q(a)$

$q_b \leftarrow q(b)$

$c \leftarrow q_a(1-u) + q_b u \quad \exp(\frac{a^2-b^2}{2})$

$\delta_x \leftarrow \infty$

$z \leftarrow 1 - u + u \quad \exp(\frac{a^2-b^2}{2})$

$x \leftarrow \sqrt{a^2 - 2\ln(z)}$

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 $\delta_x \leq \delta^*$

return x

Inversion using best standard method (Blair et al. 1976) vs our algorithm, with $r = 5$ and $\delta^* = 10^{-14}$.

a	b	u	standard method	our algorithm
10.0	12.0	0.99	10.446272896499	10.446272896855
10.0	12.0	0.30	10.035260039588	10.035260039626
20.0	22.0	0.99	20.228389499595	20.228389499595
20.0	22.0	0.30	20.017781627473	20.017781627473
30.0	32.0	0.99	30.152946658582	30.152946658582
30.0	32.0	0.30	30.011873653870	30.011873653867
40.0	42.0	0.99	—	40.114892634811
40.0	42.0	0.30	—	40.008910319783
50.0	52.0	0.99	—	50.091982066969
50.0	52.0	0.30	—	50.007130140913

Rejection methods

The fastest rejection methods for the standard normal use precomputed tables for the center of the distribution (e.g., Marsaglia's ziggurat with horizontal rectangular stripes, Chopin's method for truncated normal with about 4000 vertical stripes, etc.), and rejection with an exponential or Rayleigh proposal g in the far tail $[a, \infty)$.

Marsaglia (1964) proposed Rayleigh proposal g.

Devroye (1986) proposed exponential g with rate a.

Both have exactly the same acceptance probability!

Geweke (1991) and Robert (1995) optimized the rate of the exponential to λ that maximizes the acceptance probability. Slight improvement on acceptance, e.g., from 0.843 to 0.933 for $a = 2$, and from 0.9975 to 0.9987 for $a = 30$, but not necessarily faster, because more overhead.

When $[a, b]$ is very narrow, one can just use a uniform proposal.

When $b = \infty$, we have the OneSide case.

When $b < \infty$, one can either use a proposal that goes to infinity and reject if $X > b$ (`RejectTail`), or use a proposal truncated to $[a, b]$ (`TruncTail`). The latter gives fewer rejections, but more overhead. Worthwhile only if $\bar{\Phi}(b)/\bar{\Phi}(a) \ll 1$.

When generating a large number of variates for the same truncation interval, it may be worthwhile to precompute certain constants once for all, instead of recomputing them each time.

Computing a log is about 10 times more costly than computing a square root, and computing an exponential is 20 to 100 times more costly.

We implemented and compared the combinations of these possibilities, and we report the CPU time to generate 10^8 truncated normals, in various cases, in a Java environment, using SSJ.

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

$K_a \leftarrow 2a^2$

$q \leftarrow 1 - \exp(-(b-a)a)$

repeat

Generate $U, V \sim \mathrm{U}(0,1)$, independent

$X \leftarrow -\ln(1 - qU)$

$E \leftarrow -\ln(V)$

until $X^2 \leq K_a V$

return $a + X/a$

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

$K_a \leftarrow 2a^2$

$q \leftarrow 1 - \exp(-(b-a)a)$

repeat

Generate $U, V \sim \mathrm{U}(0,1)$, independent

$X \leftarrow -\ln(1 - qU)$

$E \leftarrow -\ln(V)$

until $X^2 \leq K_a V$

return $a + X/a$

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

$\lambda \leftarrow (a + \sqrt{a^2+4})/2$

$q \leftarrow 1 - \exp(-(b-a)\lambda)$

repeat

Generate $U, V \sim \mathrm{U}(0,1)$, independent

$X \leftarrow a - \ln(1 - qU)/\lambda$

until $V \leq \exp((X-\lambda)^2/2)$

return $a + X/a$

$X \sim \mathrm{TN}_{a,b}(0,1)$, Rayleigh proposal, truncated

$c \leftarrow a^2/2$

$q \leftarrow 1 - \exp(c - b^2/2)$

repeat

Simulate $U, V \sim \mathrm{U}(0,1)$, independently.

$X \leftarrow c - \ln(1 - qU)$

until $V^2 X \leq a$

return $X \leftarrow \sqrt{2X}$

$X \sim \mathrm{TN}_{a,b}(0,1)$, Rayleigh proposal, truncated

$c \leftarrow a^2/2$

$q \leftarrow 1 - \exp(c - b^2/2)$

repeat

Simulate $U, V \sim \mathrm{U}(0,1)$, independently.

$X \leftarrow c - \ln(1 - qU)$

until $V^2 X \leq a$

return $X \leftarrow \sqrt{2X}$

$X \sim \mathrm{TN}_{a,b}(0,1)$, Rayleigh proposal, RejectTail

$c \leftarrow a^2/2$

repeat

Simulate $U, V \sim \mathrm{U}(0,1)$, independently.

$X \leftarrow c - \ln(U)$

until $V^2 X \leq a$ and $2X \leq b * b$

return $\sqrt{2X}$

$X \sim \mathrm{TN}_{a,b}(0,1)$ with uniform proposal, truncated

repeat

Simulate $U, V \sim \mathrm{U}(0,1)$, independently.

$X \leftarrow a + (b-a)U$

until $2 \ln V \leq a^2 - X^2$

return X

$n = 10^8$ random variates in $[a, b] = [3.0, 3.1]$

Method	CPU time (seconds)	
	recompute	precompute
Generation in $[a, b)$		
ExponD	6.46	6.22
ExponDRejectTail	23.04	23.20
ExponR	16.63	9.92
ExponRRejectTail	32.40	32.40
Rayleigh	10.29	4.60
RayleighRejectTail	15.23	15.33
Uniform	4.26	4.34
InverseSSJ	15.14	8.14
InverseRightTail	31.12	7.66
Generation in $[a, \infty)$		
ExponDOneSide	6.43	6.46
RayleighOneSide	4.07	4.41
InverseSSJOneSide	18.81	8.20
InverseRightTailOneSide	18.72	7.64

$n = 10^8$ random variates in $[a, b] = [7.0, 8.0]$

Method	CPU time	
	recompute	precompute
Generation in $[a, b)$		
ExponD	11.70	6.16
ExponDRejectTail	6.04	6.08
ExponR	15.96	8.98
ExponRRejectTail	9.20	9.09
Rayleigh	9.86	4.27
RayleighRejectTail	3.91	3.99
Uniform	25.40	25.68
InverseSSJ	30.67	8.14
InverseRightTail	31.12	7.70
Generation in $[a, \infty)$		
ExponDOneSide	5.90	5.96
RayleighOneSide	3.74	4.05
InverseSSJOneSide	19.00	8.19
InverseRightTailOneSide	18.76	7.59

$n = 10^8$ **random variates in** $[a, b] = [100.0, 102.0]$

Method	CPU time	
	recompute	precompute
Generation in $[a, b)$		
ExponD	11.68	6.01
ExponDRejectTail	5.88	5.91
ExponR	15.79	8.86
ExponRRejectTail	9.13	9.02
Rayleigh	9.97	4.16
RayleighRejectTail	3.84	3.90
Uniform	650.62	656.42
InverseMillsRatio	22.31	15.97
Generation in $[a, \infty)$		
ExponDOneSide	5.77	5.82
RayleighOneSide	3.67	3.96
InverseMillsRatioOneSide	15.62	15.84

$n = 10^8$ **random variates in** $[a, b] = [100.0, 100.0001]$

Method	CPU time	
	recompute	precompute
Generation in $[a, b)$		
ExponD	12.31	6.83
ExponDRejectTail	543.80	546.58
ExponR	16.47	10.65
ExponRRejectTail	865.24	865.34
Rayleigh	10.59	5.07
RayleighRejectTail	323.08	322.41
Uniform	3.59	3.62
InverseMillsRatio	18.03	12.12
Generation in $[a, \infty)$		
ExponDOneSide	5.79	5.83
RayleighOneSide	3.66	3.99
InverseMillsRatioOneSide	15.67	15.84

Conclusion

- New accurate method for inversion in the far tail, even for $X > 10^{39}$.
- Comparisons of various rejection methods for different ranges of parameters.
  Rayleigh is generally faster than exponential, and optimizing the rate for the exponential to maximize acceptance probability is not worth.
  Over a very small interval, a uniform proposal wins.
- Java software used for these tests is available.
- To do: for rejection, construct a software that selects automatically the best combination given the interval $[a, b]$ and how many variates we want to generate.