Sampling exactly from the normal distribution

Charles Karney ⟨charles.karney@sri.com⟩
SRI International
AofA 2017, Princeton, June 20, 2017

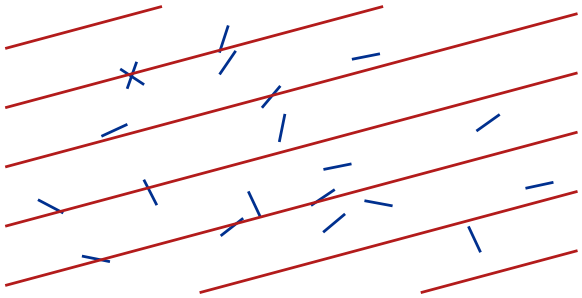
Background:

- In my day job, I've needed normal (Gaussian) deviates in Monte Carlo simulations:
  - neutral transport in plasmas for fusion (1990–2000);
  - protein-ligand binding for drug discovery (2000–2004).
- However, for nearly all scientific applications "exactness" is not required, only "accuracy"; on the other hand . . .
  - this algorithm is *much* faster than "standard" methods at arbitrary precision;
  - exactness *is* a requirement in some applications.

Defining *exact*

Two inexact methods of sampling with probability $1/\pi$:

- Throw a $1''$ needle onto $2''$ floor boards (Buffon, 1733). Does the needle cross a crack?



- Select a uniform random number $U \in (0, 1)$; test $U < 0.31830988618379067 \approx 1/\pi$.

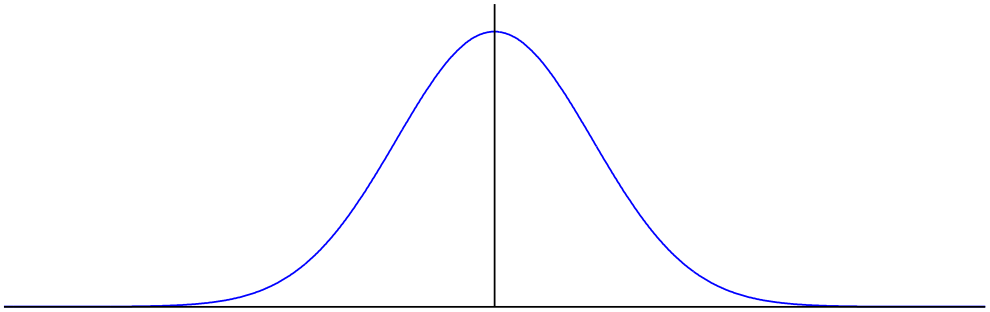
These are inexact because

- it involves physical (and therefore inexact) measurements;
- it involves floating point arithmetic (also inexact).

An exact method takes a sequence of tosses of a fair coin and succeeds with a probability of *precisely* $1/\pi$. I'll give a method which only involves counting at the end of this talk.

The normal distribution

$$\phi(x) = \frac{\exp(-\frac{1}{2}x^2)}{\sqrt{2\pi}}$$



The normal distribution is the familiar "bell-shaped" curve that models many natural phenomena: Brownian motion, IQ, hat sizes, etc. There are several well-known methods for sampling from $\phi(x)$. These are only accurate "to round-off".

We want an algorithm which *behaves* as though it were doing the following:

- take as input uniformly distributed random bits (0 and 1);
- sample a real number (with *infinite* precision) from the normal distribution;
- round this to the closest double precision float and return the result.

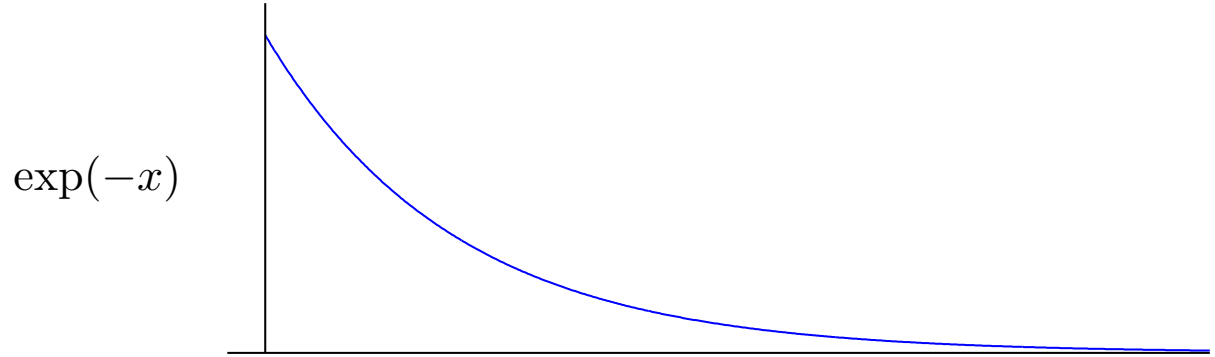
Thus the probability that 0.75 is returned should be *exactly*

$$\int_{\frac{3}{4}-2^{-54}}^{\frac{3}{4}+2^{-54}} \phi(x)\,dx.$$

It's not obvious that any such (simple) algorithm exists.

von Neumann's algorithm for the exponential distribution

A hint that there is such a solution comes from von Neumann's algorithm for the exponential distribution. This distribution models the distance a neutron travels until a fission event, the time between clicks on a Geiger counter, etc.



The standard algorithm for sampling from this distribution is to sample $U \in (0, 1)$ and return $x = -\log U$. But *"it seems objectionable to compute the transcendental function of a random number"* (von Neumann, 1949). His algorithm is:

1. Set $k \leftarrow 0$.
2. Select a uniform sample $x \in (0, 1)$.
3. Similarly select uniform samples $U_1, U_2, \ldots, U_n$; determine the largest n such that $x > U_1 > U_2 > \ldots > U_n$.
4. If n is even, return $k + x$; otherwise, set $k \leftarrow k + 1$ and go back to step 2.

von Neumann's algorithm (cont.)

Probability of $x > U_1 > U_2 > \ldots > U_n$ (given x):

- The probability that all $U_i < x$ for $i \in [1, n]$ is x^n.
- The probability that they are in the decreasing order is $1/n!$.
- The probability that the ordering holds for n but not for $n+1$ (i.e., that the sequence of n is the longest) is $x^n/n! - x^{n+1}/(n+1)!$.
- Finally, the probability that n is even is

$$1 - x \quad + \quad \frac{x^2}{2!} - \frac{x^3}{3!} \quad + \quad \frac{x^4}{4!} - \frac{x^5}{5!} \quad + \quad \ldots = \exp(-x).$$

If implemented with floating point arithmetic, this *approximately* realizes the exponential distribution and various authors (Forsythe, 1972; ...) have generalized the algorithm assuming the use of floating point arithmetic.

Knuth and Yao (1976) realized that it could be adapted to sample exactly from the exponential distribution because the only operation carried out on real numbers are comparisons and $0.31\ldots > 0.37\ldots$ can be determined by only looking at a few initial digits of the numbers. The final result of the algorithm, $k + x$, is a number, e.g., $1.34\ldots$, where some digits have been determined and the rest (denoted by ...) can be filled in with sufficient random digits to allow the result to be accurately rounded to arbitrary precision.

von Neumann's algorithm (cont.)

Remarks from von Neumann's paper:

- This method *"resembles a well known game of chance—Twenty-one, or Black Jack."* Imagine a *continuous* version of Black Jack played with cards drawn from $(0, 1)$ with "bust" defined as the total value in excess of x. For $x \leq 1$, the probability of not going bust after n cards is $x^n/n!$; the probability of going bust with an odd number of cards is $\exp(-x)$; the mean number of cards to go bust is $\exp(x)$. This was presumably a piece of trivia shared during card games in Los Alamos.
- *"It is a sad fact of life, however, that under the particular conditions of the Eniac it was slightly quicker to use a truncated power series for $\log U$ than to carry out all the discriminations."* von Neumann used the Eniac for Monte Carlo calculations in support of the H bomb work at Los Alamos. More on the timing later.
- *"The machine has in effect computed a logarithm by performing only discriminations on the relative magnitude of numbers in $(0, 1)$."* Remarkably, the algorithm sums an infinite Taylor series with a finite mean running time.

von Neumann's last statement is the most tantalizing: could a similar technique be applied to other distributions (in particular, the normal distribution)?

The algorithm must be such that the *only* operation performed on real numbers is $x < y$.

Algorithm N, sampling from the normal distribution

I posed this as a challenge to the mailing list for MPFR (a library for arbitrary precision floating point arithmetic) in January 2010. I found the solution given here in January 2012.

Here is the outline of the algorithm:

1. Select integer $k \geq 0$ with probability proportional to $\exp(-\frac{1}{2}k)$.
2. Accept k with probability $\exp\bigl(-\frac{1}{2}k(k-1)\bigr)$. The probability density of k is now $\exp(-\frac{1}{2}k^2)$.
3. Select a uniform sample $x \in (0,1)$.
4. Accept x with probability $\exp\bigl(-\frac{1}{2}x(2k+x)\bigr)$. The probability density of $[k,x]$ is now $\exp\bigl(-\frac{1}{2}(k+x)^2\bigr)$.
5. Return $\pm(k+x)$ with equal probability. The probability density of the result is now normal.

Steps 1 and 2 can be expressed in terms of Bernoulli trials with probability $\exp(-\frac{1}{2})$; this can be implemented with von Neumann's method by setting $x = \frac{1}{2}$.

Algorithm N (cont.)

The challenge is Step 4; transform this to

- Repeat $k+1$ times:
- Generate two sequences of uniform samples U_1, U_2, ..., U_n and V_1, V_2, ..., V_n; determine the largest n such that
  1. $x > U_1 > U_2 > \ldots > U_n$ (this is the von Neumann test) and
  2. $V_i < (2k+x)/(2k+2)$ for all $i \in [1, n]$.
- If n is odd, start again at the beginning of the algorithm.

Condition 1 is satisfied with probability $x^n/n!$;
condition 2 is satisfied with probability $\big((2k+x)/(2k+2)\big)^n$.
Thus the probability that n is even is

$$1 - x\frac{2k+x}{2k+2} \quad + \quad \frac{x^2}{2!}\left(\frac{2k+x}{2k+2}\right)^2 - \frac{x^3}{3!}\left(\frac{2k+x}{2k+2}\right)^3 \quad + \quad \ldots = \exp\left(-x\frac{2k+x}{2k+2}\right).$$

The probability that n is even for all $k+1$ iterations is $\exp\big(-\frac{1}{2}x(2k+x)\big)$.

The iterations are needed because $\frac{1}{2}x(2k+x)$ can be greater than 1. Dividing by $k+1$ fixes this.
Need to go down another rabbit hole to implement condition 2 without doing any real arithmetic!
See paper for details.

Properties of the Algorithm N

Because the only operations on x are inequalities, the final normal sample $\pm(k + x)$ has only a small number of digits in the fraction filled in—a form of *lazy evaluation*. For example, with base $b = 10$, let the random digits be

$$9148686 \mid 685171\ldots$$

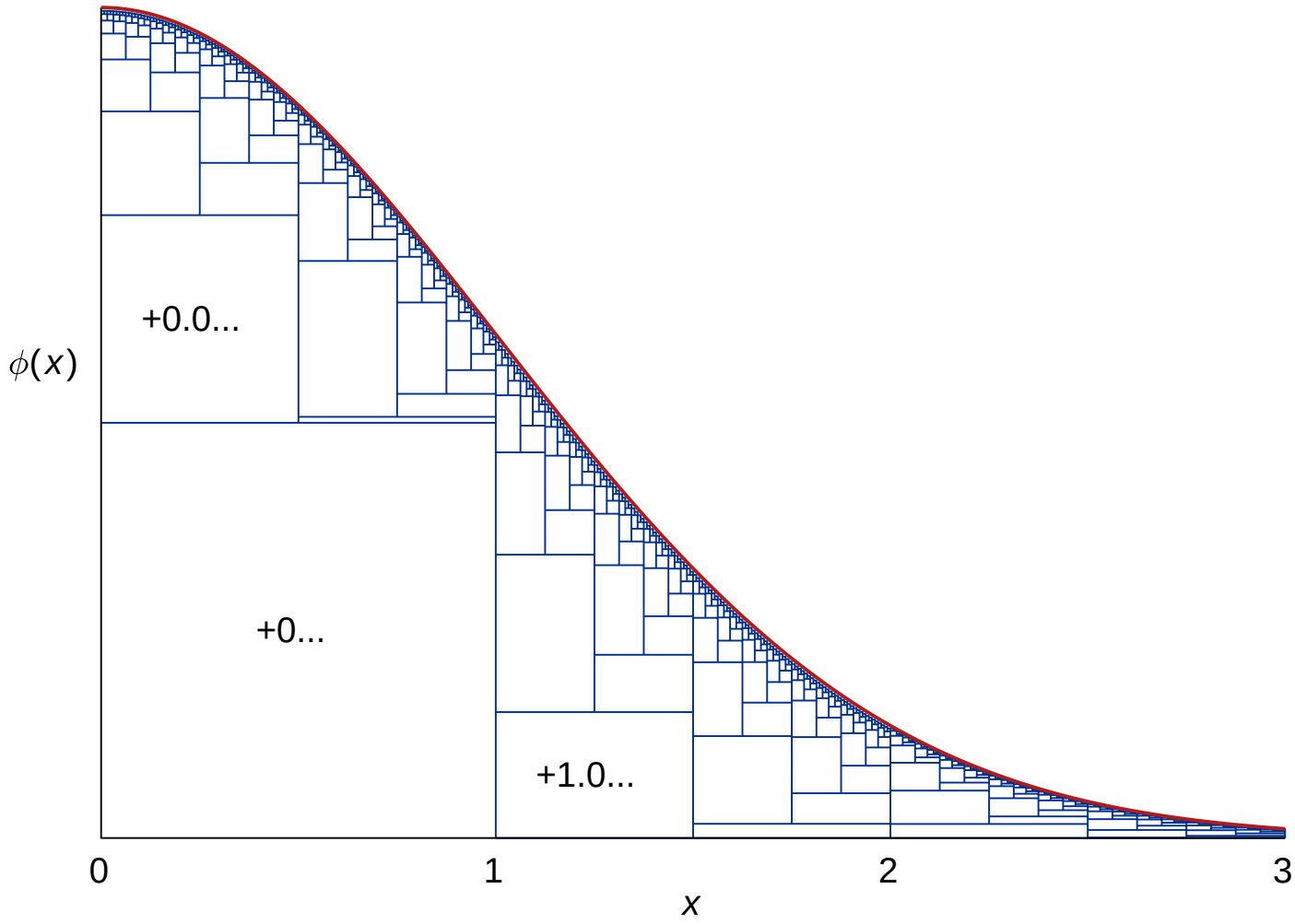
The algorithm completes after reading 7 digits to produce $+1.6\ldots$ (i.e., only one digit of the fraction has been determined. Now random digits can be copied directly to the result to give $+1.6685171\ldots$. Correctly rounded normal deviates in base 10 can be generated (and, as a bonus, you know in which direction the exact sample was rounded).

With $b = 2$ (input is random bits), the algorithm completes after consuming 30.0 bits on average and the mean number of bits in the fraction of the result is 1.6.

Define the "toll", T, of the algorithm as the difference between the number of bits consumed and the entropy (in bits) of the resulting samples.

- normal sampling: $T \approx 26.4$;
- exponential sampling (von Neumann): $T \approx 5.8$;
- exponential sampling (von Neumann with *early rejection*): $T \approx 4.0$.

Properties of the Algorithm N (cont.)



The decomposition of the normal distribution into uniform distributions.

Speed of the Algorithm N (base $b = 2^{32}$)

type	IEEE		MPFR	
method	polar	Algorithm N		polar
precision (bits)	A	B	C	D
24	0.034	0.30	0.59	2.3
32			0.64	2.4
53	0.054	0.31	0.64	2.6
64	0.057	0.37	0.64	2.8
128			0.65	3.8
256			0.68	6.2
2^{10}			0.86	20
2^{12}			1.6	130
2^{14}			4.3	1300
2^{16}			15	13000
2^{18}			59	120000
2^{20}			240	910000

Times are averages per normal deviate in μs. "Polar" is a standard method for sampling from the normal distribution. Columns A and B use IEEE floating point arithmetic. Columns C and D use the MPFR functions `mpfr_nrandom` (coming soon! in MPFR 4.0) and `mpfr_grandom`.

The average running time in Column C is offset linear in the desired precision;
it is among the fastest functions in MPFR.

Sampling from the discrete normal distribution

- Can adapt Algorithm N to sample from the discrete normal distribution,

$$\phi(i \mid \mu, \sigma) \propto \exp\left[-\frac{1}{2}\left(\frac{i-\mu}{\sigma}\right)^2\right],$$

  provided that the parameters μ and σ are rational.

- Major application is cryptography, e.g., "learning with errors" (Regev, 2009).

- In order to obtain perfect scaling, required number of bits proportional $\log_2 \sigma$ for $\sigma \gg 1$, it's necessary to modify the integer sampling of Lumbroso (2013) to return a partially sampled result—another example of lazy evaluation.

- This algorithm contradicts an assertion made in a review by Dwarakanath and Galbraith (2014): *"sampling algorithms require either high precision floating point arithmetic or very large precomputed tables"*.

- At about 100 lines of code, the algorithm is suitable for "constrained hardware" (simple integer arithmetic, limited power, limited memory). Potential issues:

  – unbounded time (and memory);

  – may subject to "side-channel" attacks.

The digital method of sampling with probability $1/\pi$

This algorithm is due to Flajolet et al. (2011):

- Perform three coin tossing experiments in which you toss a coin until you get tails, e.g., HHHHT; HHHT; HHT. Let $h_1 = 4$, $h_2 = 3$, $h_3 = 2$ be the numbers of heads tossed in each experiment.
- Compute $n = \lfloor h_1/2 \rfloor + \lfloor h_2/2 \rfloor + \mathrm{mod}(\lfloor (h_3 - 1)/3 \rfloor, 2) = 2 + 1 + 0 = 3$. Probability of n is $(6n + 1)/4^{n+1}$. Mean value of n is $\frac{11}{9}$.
- Perform three additional coin tossing experiments in which you toss a coin $2n = 6$ times, e.g., TTHHTH; HHTHH||H; THHHHH. Are the number of heads and tails equal in each experiment? $\mathrm{yes} \wedge \mathrm{no} \wedge \mathrm{no} \to \mathrm{no}$. (Here, you should give up at the ||.) Probability of success for each experiment is

$$\binom{2n}{n} \frac{1}{4^n}.$$

Working out the probability of success gives

$$\sum_{n=0}^{\infty} \frac{6n+1}{4^{n+1}} \left[\binom{2n}{n} \frac{1}{4^n} \right]^3 = \frac{1}{4} \left[1 + \frac{7}{4} \left(\frac{1}{2} \right)^3 + \frac{13}{4^2} \left(\frac{1 \cdot 3}{2 \cdot 4} \right)^3 + \frac{19}{4^3} \left(\frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \right)^3 + \ldots \right] = \frac{1}{\pi} \quad (!)$$

See Ramanujan (1914), Eq. (28). Average number of coin tosses per result is 9.6.

References

- C. F. F. Karney, ACM TOMS **42**, 3:1–14 (2016). https://arxiv.org/abs/1303.6257; C++ implementation available at http://exrandom.sourceforge.net.
- J. von Neumann, *Various techniques used in connection with random digits*, in *Monte Carlo Method* (1951).
- G. E. Forsythe, *von Neumann's comparison method for random sampling from the normal and other distributions*, Math. Comp. **26**, 817–826 (1972).
- D. E. Knuth and A. C. Yao, *The complexity of nonuniform random number generation*, in *Algorithms and Complexity* (1976).
- MPFR, *A Multiple-Precision binary Floating-point library with correct Rounding*, http://mpfr.org.
- O. Regev, *On lattices, learning with errors, random linear codes, and cryptography*, J. ACM **56**, 34:1–40 (2009).
- J. Lumbroso, *Optimal discrete uniform generation from coin flips, and applications* (2013). https://arxiv.org/abs/1304.1916
- N. C. Dwarakanath and S. D. Galbraith, *Sampling from discrete Gaussians for lattice-based cryptography on a constrained device*, AAECC **25**, 159–180 (2014).
- P. Flajolet, M. Pelletier, M. Soria, *On Buffon machines and numbers*, 22nd ACM-SIAM Symposium on Discrete Algorithms (2011). https://arxiv.org/abs/0906.5560
- S. Ramanujan, *Modular equations and approximations to* π, Quart. J. Math. **45**, 350–372 (1914).