Miscellaneous Observations on Randomization

This version of the document is dated 2023-05-25.

Peter Occil

This page should be read in conjunction with the following articles:

- **Randomization and Sampling Methods**.
- **More Random Sampling Methods**.

1 Contents

2 About This Document

This is an open-source document; for an updated version, see the source code or its rendering on GitHub. You can send comments on this document on the GitHub issues page.

My audience for this article is **computer programmers with mathematics knowledge, but little or no familiarity with calculus**.

I encourage readers to implement any of the algorithms given in this page, and report their implementation experiences. In particular, **I seek comments on the following aspects**:

- Are the algorithms in this article easy to implement? Is each algorithm written so that someone could write code for that algorithm after reading the article?
- Does this article have errors that should be corrected?
- Are there ways to make this article more useful to the target audience?

Comments on other aspects of this document are welcome.

3 Samplers for Certain Discrete Distributions

The following are exact samplers for certain *discrete distributions*, or probability distributions that take on values each mappable to a different integer.

3.1 On a Binomial Sampler

The binomial(n, p) distribution models the number of successful trials ("coin flips") out of n of them, where the trials are independent and have success probability p.

Take the following sampler of a binomial(n, 1/2) distribution, where n is even, which is equivalent to the one that appeared in Bringmann et al. (2014)[1], and adapted to be more programmer-friendly.

1. If n is less than 4, generate n unbiased random bits (each bit is zero or one with equal probability) and return their sum. Otherwise, if n is odd[2], set *ret* to the result of this algorithm with $n = n - 1$, then add an unbiased random bit's value to *ret*, then return *ret*.
2. Set m to floor(sqrt(n)) + 1.
3. (First, sample from an envelope of the binomial curve.) Generate unbiased random bits until a zero is generated this way. Set k to the number of ones generated this way.
4. Set s to an integer in [0, m) chosen uniformly at random, then set i to $k*m + s$.
5. Generate an unbiased random bit. If that bit is 0, set *ret* to $(n/2)+i$. Otherwise, set *ret* to $(n/2)-i-1$.
6. (Second, accept or reject *ret*.) If $ret < 0$ or $ret > n$, go to step 3.
7. With probability choose(n, *ret*)$*m*2^{k-n-2}$, return *ret*. Otherwise, go to step 3. (Here, choose(n, k) is a *binomial coefficient*, or the number of ways to choose k out of n labeled items.[3])

This algorithm has an acceptance rate of 1/16 regardless of the value of n. However, step 7 will generally require a growing amount of storage and time to exactly calculate the given probability as n gets larger, notably due to the inherent factorial in the binomial coefficient. The Bringmann paper suggests approximating this factorial via Spouge's approximation; however, it seems hard to do so without using floating-point arithmetic, which the paper ultimately resorts to. Alternatively, the logarithm of that probability can be calculated, then 0 minus an exponential random variate can be generated and compared with that logarithm to determine whether the step succeeds.

More specifically, step 7 can be changed as follows:

- (7.) Let p be loggamma(n+1)−loggamma(ret+1)−loggamma((n−ret)+1)+ln(m) +ln(2)*(k−n−2) (where loggamma(x) is the logarithm of the

gamma function).

- (7a.) Generate an exponential random variate with rate 1 (which is the negative natural logarithm of a uniform(0,1) random variate). Set h to 0 minus that number.
- (7b.) If h is greater than p, go to step 3. Otherwise, return *ret*. (This step can be replaced by calculating lower and upper bounds that converge to p. In that case, go to step 3 if h is greater than the upper bound, or return *ret* if h is less than the lower bound, or compute better bounds and repeat this step otherwise. See also chapter 4 of (Devroye 1986)[4].)

My implementation of loggamma and the natural logarithm (**betadist.py**) relies on so-called "constructive reals" as well as a fast converging version of Stirling's formula for the factorial's natural logarithm (Schumacher 2016)[5].

Also, according to the Bringmann paper, m can be set such that m is in the interval [sqrt(n), sqrt(n)+3], so I implement step 1 by starting with $u = 2^{\text{floor}((1+\beta(n))/2)}$, then calculating v = floor((u+floor(n/u))/2), w = u, u = v until $v \geq w$, then setting m to w + 1. Here, $\beta(n)$ = ceil(ln(n+1)/ln(2)), or alternatively the minimum number of bits needed to store n (with $\beta(0) = 0$).

> **Notes:**
>
> - A binomial(n, 1/2) random variate, where n is odd[6], can be generated by adding an unbiased random bit's value (either zero or one with equal probability) to a binomial($n-1$, 1/2) random variate.
> - As pointed out by Farach-Colton and Tsai (2015)[7], a binomial(n, p) random variate, where p is in the interval (0, 1), can be generated using binomial(n, 1/2) numbers using a procedure equivalent to the following:
>   1. Set k to 0 and *ret* to 0.
>   2. If the binary digit at position k after the point in p's binary expansion (that is, 0.bbbb... where each b is a zero or one) is 1, add a binomial(n, 1/2) random variate to *ret* and subtract the same variate from n; otherwise, set n to a binomial(n, 1/2) random variate.
>   3. If n is greater than 0, add 1 to k and go to step 2; otherwise, return *ret*. (Positions start at 0 where 0 is the most significant digit after the point, 1 is the next, etc.)

3.2 On Geometric Samplers

As used in Bringmann and Friedrich (2013)[8], a geometric(p) random variate expresses the number of failing trial before the first success, where each trial ("coin flip") is independent and has success probability p, satisfying $0 < p \le 1$.

> **Note**: The terms "geometric distribution" and "geometric random variate" have conflicting meanings in academic works.

The following algorithm is equivalent to the geometric(px/py) sampler that appeared in that paper, but adapted to be more programmer-friendly. The algorithm uses the rational number px/py, not an arbitrary real number p; some of the notes in this section indicate how to adapt the algorithm to an arbitrary p.

1. Set pn to px, k to 0, and d to 0.
2. While $pn*2 \le py$, add 1 to k and multiply pn by 2. (Equivalent to finding the largest $k \ge 0$ such that $p*2^k \le 1$. For the case when p need not be rational, enough of its binary expansion can be calculated to carry out this step accurately, but in this case any k such that p is greater than $1/(2^{k+2})$ and less than or equal to $1/(2^k)$ will suffice, as the Bringmann paper points out.)
3. With probability $(1-px/py)^{2^k}$, add 1 to d and repeat this step. (To simulate this probability, the first sub-algorithm below can be used.)
4. Generate a uniform random integer in $[0, 2^k)$, call it m, then with probability $(1-px/py)^m$, return $d*2^k+m$. Otherwise, repeat this step. (The Bringmann paper, though, suggests to simulate this probability by sampling only as many bits of m as needed to do so, rather than just generating m in one go, then using the first sub-algorithm on m. However, the implementation, given as the second sub-algorithm below, is much more complicated and is not crucial for correctness.)

The first sub-algorithm returns 1 with probability $(1-px/py)^n$, assuming that $n*px/py \le 1$. It implements the approach from the Bringmann paper by rewriting the probability using the binomial theorem. (More generally, to return 1 with probability $(1-p)^n$, it's enough to flip a coin that shows heads with probability p, n times or

until it shows heads, whichever comes first, and then return either 1 if all the flips showed tails, or 0 otherwise. See also "**Bernoulli Factory Algorithms**".)

1. Set *pnum*, *pden*, and *j* to 1, then set *r* to 0, then set *qnum* to *px*, and *qden* to *py*, then set *i* to 2.
2. If *j* is greater than *n*, go to step 5.
3. If *j* is even[9], set *pnum* to *pnum***qden* + *pden***qnum**choose(*n*,*j*). Otherwise, set *pnum* to *pnum***qden* − *pden***qnum**choose(*n*,*j*).
4. Multiply *pden* by *qden*, then multiply *qnum* by *px*, then multiply *qden* by *py*, then add 1 to *j*.
5. If *j* is less than or equal to 2 and less than or equal to *n*, go to step 2.
6. Multiply *r* by 2, then add an unbiased random bit's value (either 0 or 1 with equal probability) to *r*.
7. If *r* ≤ floor((*pnum***i*)/*pden*) − 2, return 1. If *r* ≥ floor((*pnum***i*)/*pden*) + 1, return 0. If neither is the case, multiply *i* by 2 and go to step 2.

The second sub-algorithm returns an integer *m* in [0, 2^k) with probability $(1-px/py)^m$, or −1 with the opposite probability. It assumes that 2^k**px*/*py* ≤ 1.

1. Set *r* and *m* to 0.
2. Set *b* to 0, then while *b* is less than *k*:
    1. (Sum *b*+2 summands of the binomial equivalent of the desired probability. First, append an additional bit to *m*, from most to least significant.) Generate an unbiased random bit (either 0 or 1 with equal probability). If that bit is 1, add 2^{k-b} to *m*.
    2. (Now build up the binomial probability.) Set *pnum*, *pden*, and *j* to 1, then set *qnum* to *px*, and *qden* to *py*.
    3. If *j* is greater than *m* or greater than *b* + 2, go to the sixth substep.
    4. If *j* is even[10], set *pnum* to *pnum***qden* + *pden***qnum**choose(*m*,*j*). Otherwise, set *pnum* to *pnum***qden* − *pden***qnum**choose(*m*,*j*).
    5. Multiply *pden* by *qden*, then multiply *qnum* by *px*, then multiply *qden* by *py*, then add 1 to *j*, then go to the third substep.
    6. (Now check the probability.) Multiply *r* by 2, then add an unbiased random bit's value (either 0 or 1 with equal probability) to *r*.
    7. If *r* ≤ floor((*pnum**2^b)/*pden*) − 2, add a uniform random integer

in [0, 2^{k*b}) to m and return m (and, if requested, the number $k-b-1$). If $r \geq$ floor(($pnum$*2^b)/$pden$) + 1, return -1 (and, if requested, an arbitrary value). If neither is the case, add 1 to b.

3. Add an unbiased random bit to m. (At this point, m is fully sampled.)
4. Run the first sub-algorithm with $n = m$, except in step 1 of that sub-algorithm, set r to the value of r built up by this algorithm, rather than 0, and set i to 2^k, rather than 2. If that sub-algorithm returns 1, return m (and, if requested, the number -1). Otherwise, return -1 (and, if requested, an arbitrary value).

3.2.1 Bounded Geometric Distribution

As used in the Bringmann paper, a bounded geometric(p, n) random variate is a geometric(p) random variate or n (an integer greater than 0), whichever is less. The following algorithm is equivalent to the algorithm given in that paper, but adapted to be more programmer-friendly.

1. Set pn to px, k to 0, d to 0, and $m2$ to the smallest power of 2 that is greater than n (or equivalently, 2^{bits} where $bits$ is the minimum number of bits needed to store n).
2. While pn*2 $\leq$ py, add 1 to k and multiply pn by 2.
3. With probability $(1-px/py)^{2^k}$, add 1 to d and then either return n if d*2^k is greater than or equal to $m2$, or repeat this step if less. (To simulate this probability, the first sub-algorithm above can be used.)
4. Generate a uniform random integer in [0, 2^k), call it m, then with probability $(1-px/py)^m$, return min(n, d*2^k+m). In the Bringmann paper, this step is implemented in a manner equivalent to the following (this alternative implementation, though, is not crucial for correctness):
    1. Run the second sub-algorithm above, except return two values, rather than one, in the situations given in the sub-algorithm. Call these two values m and $mbit$.
    2. If $m < 0$, go to the first substep.
    3. If $mbit \geq 0$, add 2^{mbit} times an unbiased random bit to m and subtract 1 from $mbit$. If that bit is 1 or $mbit < 0$, go to the next substep; otherwise, repeat this substep.
    4. Return n if d*2^k is greater than or equal to $m2$.
    5. Add a uniform random integer in [0, 2^{mbit+1}) to m, then return

$\min(n, d*2^k+m)$.

3.2.2 Symmetric Geometric Distribution

Samples from the symmetric geometric distribution from (Ghosh et al. 2012)[11], with parameter λ (a real number satisfying $0 < \lambda \leq 1$), in the form of an input coin with unknown probability of heads of λ.

1. Flip the input coin until it returns 1. Set n to the number of times the coin returned 0 this way.
2. Run a **Bernoulli factory algorithm for $1/(2-\lambda)$**, using the input coin. If the run returns 1, return n. Otherwise, return $-1 - n$.

This is similar to an algorithm mentioned in an appendix in Li (2021)[12], in which the input coin—

- has $\lambda = 1-\exp(-\varepsilon)$, where $\varepsilon > 0$, and
- can be built as follows using another input coin: "Run the **ExpMinus** algorithm with parameter ε, then return 1 minus the result."

The algorithm of Li generates a variate from the *discrete Laplace distribution* with parameter ε, and Canonne et al. (2020)[13] likewise gave an exact algorithm for that distribution where $\varepsilon = s/t$ is a rational number, where $s > 0$ and $t > 0$ are integers, namely an algorithm equivalent to the following:

1. Generate a uniform random integer u that satisfies $0 \leq u < t$.
2. Run the **ExpMinus** algorithm with parameter u/t. If it returns 0, go to step 1.
3. Run the **ExpMinus** algorithm with parameter 1, until a run returns 0, then set n to the number of times the algorithm returned 1 this way.
4. Set y to $\text{floor}((u+n*t)/s)$.
5. Generate an unbiased random bit (either zero or one with equal probability). If the bit is 0, return y. Otherwise, if y is 0, go to step 1. Otherwise, return $-y$.

3.3 Weighted Choice for Special Distributions

The following are algorithms to sample items whose "weights" (which are related to the probability of sampling each item) are given in a special way. They supplement the section "**Weighted Choice**" in my article "Randomization and Sampling Methods".

3.3.1 Weighted Choice with Weights Written as an Integer and Fraction

Suppose there is a list called *weights*. This is a list of n weights, with labels starting at 0 and ending at $n-1$.

Each weight—

1. can store an integer part m and have ν represent a "coin" that implements an algorithm that returns 1 (or outputs heads) with probability exactly equal to the fractional part ν ($m \geq 0$, and $0 \leq \nu \leq 1$), or
2. can store a **partially-sampled random number** (PSRN), with the integer part equal to m and the fractional part equal to ν ($m \geq 0$, and $0 \leq \nu \leq 1$), or
3. can store a rational number x/y, where $x \geq 0$ and $y > 0$ are integers, such that $m = \text{floor}(x/y)$ and $\nu = x/y - m$.

Given this list of weights, the following algorithm chooses an integer in $[0, n)$ with probability proportional to its weight.

1. Create an empty list, then for each weight starting with weight 0, append the weight's integer part (m) plus 1 to that list. For example, if the weights are PSRNs written as [2.22...,0.001...,1.3...], in that order, the list will be [3, 1, 2], corresponding to integers 0, 1, and 2, in that order. Call the list just created the *rounded weights list*.

2. Choose an integer i with probability proportional to the weights in the rounded weights list. This can be done, for example, by taking the result of **WeightedChoice**(*list*), where *list* is the rounded weights list and **WeightedChoice** is given in "**Randomization and Sampling Methods**". Let w be the original weight for integer i, and let rw be the rounded weight for integer i in the rounded weights list.

3. Generate j, a uniform random integer that is 0 or greater and less than rw. If j is less than $rw-1$, return i. Otherwise:

- If w is written as in case 1, above, flip the "coin" represented by ν (the weight's fractional part). If it returns 1, return i. Otherwise, go to step 2.
- If w is written as in case 2, run **SampleGeometricBag** on the PSRN. If the result is 1, return i. Otherwise, go to step 2.
- If w is written as in case 3, let $r = \text{rem}(x, y) = x - \text{floor}(x/y)*y$, then with probability r/y, return i. (For example, generate z, a uniform random integer satisfying $0 \le z < y$, then if $z < r$, return i.) Otherwise, go to step 2.

3.3.2 Distributions with nowhere increasing or nowhere decreasing weights

An algorithm for sampling an integer in the interval $[a, b)$ with probability proportional to weights listed in *nowhere increasing* order (example: [10, 3, 2, 1, 1] when $a = 0$ and $b = 5$) can be implemented as follows (Chewi et al. 2022)[14]. It has a logarithmic time complexity in terms of setup and sampling.

- Setup: Let $w[i]$ be the weight for integer i (with i starting at a).
  1. (Envelope weights.) Build a list q as follows: The first item is $w[a]$, then set j to 1, then while $j < b-a$, append $w[a + j]$ and multiply j by 2. The list q's items should be rational numbers that equal the true values, if possible, or overestimate them if not.
  2. (Envelope chunk weights.) Build a list r as follows: The first item is $q[0]$, then set j to 1 and m to 1, then while $j < b-a$, append $q[m]*\min((b-a) - j, j)$ and multiply j by 2 and add 1 to m.
  3. (Start and end points of each chunk.) Build a list D as follows: The first item is the list $[a, a+1]$, then set j to 1, then while $j < n$, append the list $[j, j + \min((b-a) - j, j)]$ and multiply j by 2.
- Sampling:
  1. Choose an integer in $[0, s)$ with probability proportional to the weights in r, where s is the number of items in r. Call the chosen integer k.
  2. Set x to an integer chosen uniformly at random such that x is greater than or equal to $D[k][0]$ and is less than $D[k][1]$.
  3. With probability $w[x] / q[k]$, return x. Otherwise, go to step 1.

For *nowhere decreasing* rather than nowhere increasing weights, the algorithm is as follows instead:

- Setup: Let $w[i]$ be the weight for integer i (with i starting at a).
  1. (Envelope weights.) Build a list q as follows: The first item is $w[b-1]$, then set j to 1, then while $j < (b-a)$, append $w[b-1-j]$ and multiply j by 2. The list q's items should be rational numbers that equal the true values, if possible, or overestimate them if not.
  2. (Envelope chunk weights.) Build a list r as given in step 2 of the previous algorithm's setup.
  3. (Start and end points of each chunk.) Build a list D as follows: The first item is the list $[b-1, b]$, then set j to 1, then while $j < (b-a)$, append the list $[(b-j) - \min((b-a) - j, j), b-j]$ and multiply j by 2.
- The sampling is the same as for the previous algorithm.

> **Notes:**
>
> 1. The weights can be base-β logarithms, especially since logarithms preserve order, but in this case the algorithm requires changes. In the setup step 2, replace "$q[m]$*min($(b-a)$" with "$q[m]$+ln(min($(b-a)$)/ln(β)" (which is generally inexact unless β is 2); in sampling step 1, use an algorithm that takes base-β logarithms as weights; and replace sampling step 3 with "Generate an exponential random variate with rate ln(β) (that is, the variate is E/ln(β) where E is an exponential random variate with rate 1). If that variate is greater than $q[k]$ minus $w[x]$, return x. Otherwise, go to step 1."
>
>   Applying these modifications to this section's algorithms can introduce numerical errors unless care is taken (see note 2). The same is true for running the unmodified algorithms with weights that are not rational numbers.
> 2. If an algorithm will operate on potentially irrational numbers, then to avoid numerical errors, it should store and operate on real numbers in the form of constructive reals or recursive reals (see, e.g., Boehm 1987[15], 2020[16]), or in the form of partially-sampled random numbers (PSRNs) together with algorithms with **desirable properties for PSRN samplers**.

3.3.3 Unimodal distributions of weights

The following is an algorithm for sampling an integer in the interval [a, b) with probability proportional to a *unimodal distribution* of weights (that is, nowhere decreasing on the left and nowhere increasing on the right) (Chewi et al. 2022)[17]. It assumes the mode (the point with the highest weight) is known. An example is [1, 3, 9, 4, 4] when $a = 0$ and $b = 5$, and the *mode* is 2, which corresponds to the weight 9. It has a logarithmic time complexity in terms of setup and sampling.

- Setup:
  1. Find the point with the highest weight, such as via binary search. Call this point *mode*.
  2. Run the setup for *nowhere decreasing* weights on the interval [a, *mode*), then run the setup for *nowhere increasing* weights on the interval [*mode*, b). Both setups are described in the previous section. Then, concatenate the two q lists into one, the two r lists into one, and the two D lists into one.
- The sampling is the same as for the algorithms in the previous section.

3.3.4 Weighted Choice with Log Probabilities

Huijben et al. (2022)[18] reviews the Gumbel max trick and Gumbel softmax distributions.

> **Note**: Because these algorithms involve adding one real number to another and calculating exp of a real number, they can introduce numerical errors unless care is taken (see note 2 in "Distributions with nowhere increasing or nowhere decreasing weights", above).

Weighted choice with the Gumbel max trick. Let C>0 be an unknown number. Then, given—

- a vector of the form [p_0, p_1, ..., p_n], where p_i is a so-called "unnormalized log probability" of the form $\ln(x)+C$ (where C is a constant and x is the probability of getting i),

an integer in the closed interval [0, n] can be sampled as follows:

1. ("Gumbel".) For each p_i, generate a "Gumbel variate" G, then set q_i to p_i+G. (A so-called "Gumbel variate" is distributed as $-\ln(-\ln(U))$, where U is a uniform random variate greater than 0

and less than 1.[19])

2. ("Max".) Return the integer i corresponding to the highest q_i value.

> **Note:** "Gumbel top k sampling" samples k items according to their "unnormalized log probabilities" (see Fig. 7 of Huijben et al. (2022)[20]); this sampling works by doing step 1, then choosing the k integers corresponding to the k highest q_i values. With this sampling, though, the probability of getting i (if the plain Gumbel max trick were used) is not necessarily the probability that i is included in the k-item sample (Tillé 2023)[21].

Weighted choice with the Gumbel softmax trick. Given a vector described above as well as a "temperature" parameter $\lambda > 0$, a "continuous relaxation" or "concrete distribution" (which transforms the vector to a new one) can be sampled as follows:

1. ("Gumbel".) For each p_i, generate a "Gumbel variate" G, then set q_i to p_i+G.
2. ("Softmax".) For each q_i, set it to $\exp(q_i/\lambda)$.
3. Set d to the sum of all values of q_i.
4. For each q_i, divide it by d.

The algorithm's result is a vector q, which can be used only once to sample i with probability proportional to q_i (which is not a "log probability"). (In this case, steps 3 and 4 above can be omitted if that sampling method can work with weights that need not sum to 1.)

3.4 Bernoulli Distribution for Cumulative Distribution Functions

Suppose a real number z is given (which might be a partially-sampled random number [PSRN] or a rational number). If a probability distribution—

- has a probability density function (PDF) (as with the normal or exponential distribution), and
- has an arbitrary-precision sampler that returns a PSRN X,

then it's possible to generate 1 with the same probability as the sampler returns an X that is less than or equal to z, as follows:

1. Run the arbitrary-precision sampler to generate X, a uniform

PSRN.
2. Run **RandLess** (if z is a PSRN) or **RandLessThanReal** (if z is a real number) with parameters X and z, in that order, and return the result.

Specifically, the probability of returning 1 is the *cumulative distribution function* (CDF) for the distribution of X.

> **Notes:**
>
> 1. Although step 2 of the algorithm checks whether X is merely less than z, this is still correct; because the distribution of X has a PDF, X is less than z with the same probability as X is less than or equal to z.
> 2. All probability distributions have a CDF, not just those with a PDF, but also discrete ones such as Poisson or binomial.

3.5 Bit Vectors with Random Bit Flips

Chakraborty and Vardeman (2021)[22] describes distributions of bit vectors with a random number of bit flips. Given three parameters — μ is a p-item vector (list) with only zeros and/or ones; p is the size of μ; and α is a spread parameter greater than 0 and less than 1 — do the following to generate such a vector:

1. Generate a random integer c in the interval [0, p] in some way. (c need not be uniformly distributed. This is the number of bit flips.)
2. Create a p-item list ν, where the first c items are ones and the rest are zeros. **Shuffle** the list.
3. Create a copy of μ, call it M. Then for each i where $\nu[i] = 1$, set $M[i]$ to $1 - M[i]$. Then return M.

The paper describes two ways to establish the weights for c in step 1 (there are others as well):

- Generate c with probability proportional to the following weights: [α^0, α^1, ..., α^p]. (Since each weight is 1 or less, this can be implemented as follows, for example. Generate a uniform random integer in [0, p], call it d, then flip a coin that shows heads with probability α, d times, then either return d if d is 0 or all the flips are heads, or repeat this process otherwise.)
- Generate c with probability proportional to the following weights: [α^0*choose(p,0), α^1*choose(p,1), ..., α^p*choose(p,p)]. (Since the

sum of weights is no more than 2^p, each weight can be divided by 2^p to get weights that are 1 or less, so that this can be implemented as follows, for example. Generate a uniform random integer in [0, p], call it d, then flip a coin that shows heads with probability α, d times, and a coin that shows heads with probability choose(p, d)/2^p once, then either return d if all the flips are heads, or repeat this process otherwise. Note that the probability choose(p, d)/2^p is simple to simulate for being a rational number.)

3.6 Log-Uniform Distribution

Samples from the so-called "log uniform distribution" as used by the Abseil programming library. This algorithm takes a maximum mx and a logarithmic base b, and chooses an integer in [0, mx] such that two values are chosen with the same probability if their base-b logarithms are equal in their integer parts (which roughly means that lower numbers occur with an exponentially greater probability). Although this algorithm works, in principle, for every $b > 0$, Abseil supports only integer bases b.

1. Let L be ceil(ln(mx+1)/ln(b)). Choose a uniform random integer in the closed interval [0, L], call it u.
2. If u is 0, return 0.
3. Set st to min(mx, ceil(b^{u-1})).
4. Set en to min(mx, ceil(b^u) − 1).
5. Choose a uniform random integer in the closed interval [st, en], and return it.

4 Sampling Unbounded Monotone Density Functions

This section shows a preprocessing algorithm to generate a random variate in the closed interval [0, 1] from a distribution whose probability density function (PDF)—

- is continuous in the interval [0, 1],
- is strictly decreasing in [0, 1], and
- has an unbounded peak at 0.

The trick here is to sample the peak in such a way that the result is either forced to be 0 or forced to belong to the bounded part of the PDF. This algorithm does not require the area under the curve of the PDF in [0, 1] to be 1; in other words, this algorithm works even if the PDF is known up to a normalizing constant. The algorithm is as follows.

1. Set i to 1.
2. Calculate the cumulative probability of the interval [0, 2^{-i}] and that of [0, $2^{-(i-1)}$], call them p and t, respectively.
3. With probability p/t, add 1 to i and go to step 2. (Alternatively, if i is equal to or higher than the desired number of fractional bits in the result, return 0 instead of adding 1 and going to step 2.)
4. At this point, the PDF at [2^{-i}, $2^{-(i-1)}$) is less than or equal to a finite number, so sample a random variate in this interval using any appropriate algorithm, including rejection sampling. Because the PDF is strictly decreasing, the peak of the PDF at this interval is located at 2^{-i}, so that rejection sampling becomes trivial.

It is relatively straightforward to adapt this algorithm for strictly increasing PDFs with the unbounded peak at 1, or to PDFs with a different domain than [0, 1].

This algorithm is similar to the "inversion–rejection" algorithm mentioned in section 4.4 of chapter 7 of Devroye's *Non-Uniform Random Variate Generation* (1986)[23]. I was unaware of that algorithm at the time I started writing the text that became this section (Jul. 25, 2020). The difference here is that it assumes the whole distribution has support [0, 1] ("support" is defined later), while the algorithm presented in this article doesn't make that assumption (for example, the interval [0, 1] can cover only part of the distribution's support).

By the way, this algorithm arose while trying to devise an algorithm that can generate an integer power of a uniform random variate, with arbitrary precision, without actually calculating that power (a naïve calculation that is merely an approximation and usually introduces bias); for more information, see the article on **partially-sampled random numbers**. Even so, the algorithm I have come up with in this note may be of independent interest.

In the case of powers of a uniform random variate between 0 and 1, call the variate X, namely X^n, the ratio p/t in this algorithm has a very simple form, namely $(1/2)^{1/n}$. Note that this formula is the same regardless of i. (To return 1 with probability $(1/2)^{1/n}$, the algorithm for $\boldsymbol{(a/b)^z}$ in "**Bernoulli Factory Algorithms**" can be used with a=1, b=2, and z=1/n.) This is found by taking the PDF $f(x) = x^{1/n}/(x * n)$ and finding the appropriate p/t ratios by integrating f over the two intervals mentioned in step 2 of the algorithm.

5 Certain Families of Distributions

This section is a note on certain families of univariate (one-variable) probability distributions, with emphasis on generating random variates from them. Some of these families are described in Ahmad et al. (2019)[24], Jones (2015)[25].

The following mathematical definitions are used:

- A probability distribution's *quantile function* (also known as *inverse cumulative distribution function* or *inverse CDF*) is a nowhere decreasing function that maps uniform random variates greater than 0 and less than 1 to numbers that follow the distribution.
- A probability distribution's *support* is the set of values the distribution can take on, plus that set's endpoints. For example, the beta distribution's support is the closed interval [0, 1], and the normal distribution's support is the entire real line.
- The *zero-truncated Poisson* distribution: To generate a random variate that follows this distribution (with parameter $\lambda > 0$), generate random variates from the **Poisson distribution** with parameter λ until a variate other than 0 is generated this way, then take the last generated variate.

G families. In general, families of the form "X-G" (such as "beta-G" (Eugene et al., 2002)[26]) use two distributions, X and G, where—

- X is a probability distribution whose support is the closed interval [0, 1], and
- G is a probability distribution that should have an easy-to-compute quantile function.

The following algorithm samples a random variate following a distribution from this kind of family:

1. Generate a random variate that follows the distribution X. (Or generate a uniform **partially-sampled random number (PSRN)** that follows the distribution X.) Call the number x.
2. Calculate the quantile for G of x, and return that quantile. (If x is a uniform PSRN, see "Random Variate Generation via Quantiles", later.)

Certain special cases of the "X-G" families, such as the following, use a specially designed distribution for X:

- The *exp-G* family (Barreto-Souza and Simas 2010/2013)[27], where X is an exponential distribution, truncated to the interval [0, 1], with parameter $\lambda \geq 0$; step 1 is modified to read: "Generate U, a uniform random variate between 0 and 1, then set x to $-\ln((\exp(-\lambda)-1)*U + 1)/\lambda$ if λ != 0, and U otherwise." (The *alpha power* or *alpha power transformed* family (Mahdavi and Kundu 2017)[28] uses the same distribution for X, but with $\lambda=-\ln(\alpha)$ where α is in (0, 1]; see also Jones (2018)[29].)
- One family uses a shape parameter $a > 0$; step 1 is modified to read: "Generate u, a uniform random variate between 0 and 1, then set x to $u^{1/a}$." This family is mentioned in Lehmann (1953)[30], Durrans (1992)[31], and Mudholkar and Srivastava (1993)[32], which called it *exponentiated*.
- The *transmuted-G* family (Shaw and Buckley 2007)[33]. The family uses a shape parameter η satisfying $-1 \leq \eta \leq 1$; step 1 is modified to read: "Generate a piecewise linear random variate between 0 and 1 with weight $1-\eta$ at 0 and weight $1+\eta$ at 1, call the number x. (It can be generated as follows, see also (Devroye 1986, p. 71-72)[34]: With probability $\min(1-\eta, 1+\eta)$, generate x, a uniform random variate between 0 and 1. Otherwise, generate two uniform random variates between 0 and 1, set x to the higher of the two, then if η is less than 0, set x to $1-x$.)". ((Granzotto et al. 2017)[35] mentions the same distribution, but with a parameter $\lambda = \eta + 1$ satisfying $0 \leq \lambda \leq 2$.)
- A *cubic rank transmuted* distribution (Granzotto et al. 2017)[36] uses parameters λ_0 and λ_1 in the interval [0, 1]; step 1 is modified to read: "Generate three uniform random variates between 0 and 1, then sort them in ascending order. Then, choose 1, 2, or 3 with

probability proportional to these weights: [λ_0, λ_1, $3-\lambda_0-\lambda_1$]. Then set x to the first, second, or third variate if 1, 2, or 3 is chosen this way, respectively."

- Biweight distribution (Al-Khazaleh and Alzoubi 2021)[37]: Step 1 is modified to read: "Generate a uniform random variate x in [0, 1], then with probability $(1-x^2)^2$, go to the next step. Otherwise, repeat this process."; or "Create a uniform PSRN x with positive sign and integer part 0, then run **SampleGeometricBag** on that PSRN four times. If the first two results are not both 1 and if the last two results are not both 1, go to the next step; otherwise, repeat this process."

Transformed–transformer family. In fact, the "X-G" families are a special case of the so-called "transformed–transformer" family of distributions introduced by Alzaatreh et al. (2013)[38] that uses two distributions, X and G, where X (the "transformed") is an arbitrary distribution with a probability density function; G (the "transformer") is a distribution with an easy-to-compute quantile function; and W is a nowhere decreasing function that, among other conditions, maps a number in the closed interval [0, 1] to a number with the same support as X. The following algorithm samples a random variate from this kind of family:

1. Generate a random variate that follows the distribution X. (Or generate a uniform PSRN that follows X.) Call the number x.
2. Calculate $w = W^{-1}(x)$ (where $W^{-1}(.)$ is the inverse of W), then calculate the quantile for G of w and return that quantile. (If x is a uniform PSRN, see "Random Variate Generation via Quantiles", later.)

The following are special cases of the "transformed–transformer" family:

- The "T-R{Y}" family (Aljarrah et al., 2014)[39], in which T is an arbitrary distribution with a PDF (X in the algorithm above), R is a distribution with an easy-to-compute quantile function (G in the algorithm above), and W is the quantile function for the distribution Y, whose support must contain the support of T (so that $W^{-1}(x)$ is the cumulative distribution function for Y, or the probability that a Y-distributed number is x or less).
- Several versions of W have been proposed for the case when distribution X's support is [0, ∞), such as the Rayleigh and gamma

distributions. They include:

- $W(x) = -\ln(1-x)$ ($W^{-1}(x) = 1-\exp(-x)$). Suggested in the original paper by Alzaatreh et al.
- $W(x) = x/(1-x)$ ($W^{-1}(x) = x/(1+x)$). Suggested in the original paper by Alzaatreh et al. This choice forms the so-called "odd X G" family, and one example is the "odd log-logistic G" family (Gleaton and Lynch 2006)[40].

> **Example:** For the "generalized odd gamma-G" family (Hosseini et al. 2018)[41], X is the gamma(α) distribution, $W^{-1}(x) = (x/(1+x))^{1/\beta}$, G is arbitrary, $\alpha>0$, and $\beta>0$.

A family very similar to the "transformed–transformer" family uses a *decreasing* W.

- When distribution X's support is $[0, \infty)$, one such W that has been proposed is $W(x) = -\ln(x)$ ($W^{-1}(x) = \exp(-x)$; examples include the "Rayleigh-G" family or "Rayleigh–Rayleigh" distribution (Al Noor and Assi 2020)[42], as well as the "generalized gamma-G" family, where "generalized gamma" refers to the Stacy distribution (Boshi et al. 2020)[43]).

Minimums, maximums, and sums. Some distributions are described as a minimum, maximum, or sum of N independent random variates distributed as X, where $N \geq 1$ is an independent integer distributed as the discrete distribution Y.

- Tahir and Cordeiro (2016)[44] calls a distribution of minimums a *compound distribution*, and a distribution of maximums a *complementary compound distribution*.
- Pérez-Casany et al. (2016)[45] calls a distribution of minimums or of maximums a *random-stopped extreme distribution*.
- Let S be a sum of N variates as described above. Then Amponsah et al. (2021)[46] describe the distribution of (S, N), a two-variable random variate often called an *episode*.
- A distribution of sums can be called a *stopped-sum distribution* (Johnson et al. 2005)[47]. (In this case, N can be 0 so that $N \geq 0$ is an integer distributed as Y.)

A variate following a distribution of minimums or of maximums can be generated as follows (Duarte-López et al. 2021)[48]:

1. Generate a uniform random variate between 0 and 1. (Or generate a uniform PSRN with integer part 0, positive sign, and empty fractional part.) Call the number x.
2. For minimums, calculate the quantile for X of $1-W^{-1}(x)$ (where $W^{-1}(.)$ is the inverse of Y's probability generating function), and return that quantile.[49] (If x is a uniform PSRN, see "Random Variate Generation via Quantiles", later. Y's probability generating function is $W(z) = a[0]*z^0 + a[1]*z^1 + ...$, where $0 < z < 1$ and $a[i]$ is the probability that a Y-distributed variate equals i. See example below.)
3. For maximums, calculate the quantile for X of $W^{-1}(x)$, and return that quantile.

Examples:

This distribution:	Is a distribution of:	Where X is:	And Y is:
Geometric zero-truncated Poisson (Akdoğan et al., 2020)[50].	Maximums.	1 plus the number of failures before the first success, with each success having the same probability.	Zero-truncated Poisson.
GMDP(α, β, δ, p) (Amponsah et al. 2021)[51] (α>0, β>0, δ>0, 0<p<1).	(S, N) episodes.	Gamma(α) variate divided by β.	Discrete Pareto(δ, p) (see "Certain Distributions").
Bivariate gamma geometric(α, β, p) (Barreto-Souza 2012)[52] (α>0, β>0, 0<p<1).	(S, N) episodes.	Gamma(α) variate divided by β.	1 plus the number of failures before the first success, with each success having probability p.

Exponential Poisson (Kuş 2007)[53].	Minimums.	Exponential.	Zero-truncated Poisson.
Poisson exponential (Cancho et al. 2011)[54].	Maximums.	Exponential.	Zero-truncated Poisson.
Right-truncated Weibull(a, b, c) (Jodrá 2020)[55] (a, b, and c are greater than 0).	Minimums.	Power function(b, c).	Zero-truncated Poisson(a*c^b).

Example: If Y is zero-truncated Poisson with parameter λ, its probability generating function is $W(z)=\frac{1-\exp(z\lambda)}{1-\exp(\lambda)}$, and solving for x leads to its inverse: $W^{-1}(x)=\ln(1-x+x\times\exp(\lambda))/\lambda$.

Note: Bivariate exponential geometric (Barreto-Souza 2012)[56] is a special case of bivariate gamma geometric with α=1.

Inverse distributions. An *inverse X distribution* (or *inverted X distribution*) is generally the distribution of 1 divided by a random variate distributed as X. For example, an *inverse exponential* random variate (Keller and Kamath 1982)[57] is 1 divided by an exponential random variate with rate 1 (and so is distributed as $-1/\ln(U)$ where U is a uniform random variate between 0 and 1) and may be multiplied by a parameter $\theta > 0$.

Weighted distributions. A *weighted X distribution* uses a distribution X and a weight function $w(x)$ whose values lie in [0, 1] everywhere in X's support. The following algorithm samples from a weighted distribution (see also (Devroye 1986, p. 47)[58]):

1. Generate a random variate that follows the distribution X. (Or generate a uniform PSRN that follows X.) Call the number x.
2. With probability $w(x)$, return x. Otherwise, go to step 1.

Some weighted distributions allow any weight function $w(x)$ whose values are nonnegative everywhere in X's support (Rao 1985)[59]. (If $w(x) = x$, the distribution is often called a *length-biased* or *size-biased distribution*; if $w(x) = x^2$, *area-biased*.) Their probability density functions (PDFs) are proportional to the original PDFs multiplied by $w(x)$.

Inflated distributions. To generate an *inflated X* (also called *c-inflated X* or *c-adjusted X*) random variate with parameters c and α, generate—

- c with probability α, and
- a random variate distributed as X otherwise.

For example, a *zero-inflated beta* random variate is 0 with probability α and a beta random variate otherwise (the parameter c is 0) (Ospina and Ferrari 2010)[60] A zero-and-one inflated X distribution is 0 or 1 with probability α and distributed as X otherwise. For example, to generate a *zero-and-one-inflated unit Lindley* random variate (with parameters α, θ, and p) (Chakraborty and Bhattacharjee 2021)[61]:

1. With probability α, return a number that is 0 with probability p and 1 otherwise.
2. Generate a unit Lindley(θ) random variate, that is, generate $x/(1+x)$ where x is a **Lindley(θ) random variate**.

> **Note:** A zero-inflated X distribution where X takes on 0 with probability 0 is also called a *hurdle distribution* (Mullahy 1986)[62].

Unit distributions. To generate a *unit X* random variate (where X is a distribution whose support is the positive real line), generate a random variate distributed as X, call it x, then return $\exp(-x)$ or $1-\exp(-x)$ (also known as "Type I" or "Type II", respectively). For example, a unit gamma distribution is also known as the *Grassia distribution* (Grassia 1977)[63].

CDF–quantile family. Given two distributions X and Y (which can be the same), a location parameter $\mu \geq 0$, and a dispersion parameter $\sigma>0$, a variate from this family of distributions can be generated as follows (Smithson and Shou 2019)[64]:

1. Generate a random variate that follows the distribution X. (Or

generate a uniform PSRN that follows X.) Call the number x.
2. If distribution X's support is the positive real line, calculate x as $\ln(x)$.
3. Calculate z as $\mu+\sigma*x$.
4. If distribution Y's support is the positive real line, calculate z as $\exp(z)$.
5. Return $H(z)$.

In this algorithm:

- X and Y are distributions that each have support on either the whole real line or the positive real line. However, the book intends X to have an easy-to-compute quantile function.
- $H(z)$ is Y's *cumulative distribution function*, or the probability that a Y-distributed random variate is z or less. The book likewise intends H to be easy to compute.

> **Note:** An important property for use in statistical estimation is *identifiability*. A family of distributions is *identifiable* if it has the property that if two parameter vectors (θ_1 and θ_2) determine the same distribution, then θ_1 must equal θ_2.

6 Certain Distributions

In the table below, U is a uniform random variate between 0 and 1, and all random variates are independently generated.

This distribution:	Is distributed as:	And uses these parameters:
Power function(a, c).	$c*U^{1/a}$.	$a > 0$, $c > 0$.
Lehmann Weibull($a1$, $a2$, β) (Elgohari and Yousof 2020)[65].	$(\ln(1/U)/\beta)^{1/a1}/a2$ or $(E/\beta)^{1/a1}/a2$	$a1$, $a2$, $\beta > 0$. E is an exponential random variate with rate 1.
Marshall–Olkin(α) (Marshall and Olkin 1997)[66]	$(1-U)/(U*(\alpha-1) + 1)$.	α in [0, 1].

Lomax(α).	$(1-U)^{-1/\alpha}-1$.	$\alpha > 0$.
Power Lomax(α, β) (Rady et al. 2016)[67].	$L^{1/\beta}$	$\beta > 0$; L is Lomax(α).
Topp–Leone(α).	$1-\text{sqrt}(1-U^{1/\alpha})$.	$\alpha > 0$.
Bell–Touchard(a, b) (Castellares et al. 2020)[68].	Sum of N zero-truncated Poisson(a) random variates, where N is Poisson with parameter $b*\exp(a)-b$.[69]	$a>0$, $b>0$.
Bell(a) (Castellares et al. 2020)[70].	Bell–Touchard(a, 0).	$a>0$.
Discrete Pareto(δ, p) (Buddana and Kozubowski 2014)[71]	1 plus the number of failures before the first success, with each success having probability $1-\exp(-Z)$, where Z is a gamma($1/\delta$) variate times $-\delta*\ln(1-p)$.	$\delta > 0$, and $0<p<1$.
Neyman type A(δ, τ) (Batsidis and Lemonte 2021)[72]	Bell–Touchard(τ, $\delta*\exp(-\tau)$).	$\delta>0$, $\tau>0$.
Gamma exponential (Kudryavtsev 2019)[73].	$\delta*\text{Gamma}(t)^{1/\nu}/\text{Gamma}(s)^{r/\nu}$, where Gamma($x$) is a gamma($x$) variate.	$0 \le r < 1$; $\nu \neq 0$; $s>0$; $t>0$; $\delta>0$.
Extended xgamma (Saha et al. 2019)[74]	Gamma(α + c) variate divided by θ, where c is either 0 with probability $\theta/(\theta+\beta)$, or 2 otherwise.	$\theta>0$, $\alpha>0$, $\beta \ge 0$.
Generalized Pareto(a, b) (McNeil et al. 2010)[75]	$a*((1/(1-U))^b-1)/b$.	$a>0$; $b>0$.

Skew symmetric or symmetry-modulated (Azzalini and Capitanio 2003)[76], (Azzalini 2022)[77].	Z if $T \leq w(Z)$, or $-Z$ otherwise.	Z follows a symmetric distribution around 0; T follows a symmetric distribution (not necessarily around 0). $w(x)$ satisfies $-w(x) = w(-x)$.
Skew normal (Azzalini 1985)[78].	Skew symmetric with Z and T both separate Normal(0, 1) variates, and $w(x) = x*\alpha$.	α is a real number.
Logarithmic skew normal (Gómez-Déniz et al. 2020)[79]	exp(SNE(λ,λ)*σ+μ).	μ and λ are real numbers; $\sigma > 0$. SNE is described later.
Tilted beta binomial (Hahn 2022)[80]	Binomial(n, Tilted-beta(θ, ν, α, β)) variate.	$0 \leq \theta \leq 1$; $0 \leq \nu \leq 1$; α>0, β>0; $n \geq 0$ is an integer.
Two-piece distribution (Rubio and Steel 2020)[81].	μ − abs(Z)*$sigma1$ with probability $sigma1$/($sigma1$+$sigma2$), or μ + abs(Z)*$sigma2$ otherwise.	μ is a real number; $sigma1$>0; $sigma2$>0; Z follows a symmetric distribution around 0.
Asymmetric generalized Gaussian (Tesei and Regazzoni 1996)[82]	Two-piece distribution where Z is exponential-power(α).	α>0; μ is a real number; $sigma1$>0; $sigma2$>0.

This distribution:	**Can be sampled with the following algorithms:**	**And uses these parameters:**
Offset-symmetric Gaussian	(1) Generate an unbiased random bit b (either 0 or 1 with equal probability); (2) generate Y, a Normal(0, σ) random variate	m>0; σ>0.

(Sadeghi and Korki 2021)[83]	(standard deviation σ), and if $Y < m$, repeat this step; (3) return $(Y - m)*(b*2 - 1)$.	
Generalized skew normal (SNE(λ,ξ)) (Henze 1986)[84]	**First algorithm:** (1) Generate Y and Z, two Normal(0,1) variates; (2) if $Z<Y*\lambda+\xi$, return Y; else go to 1. **Second algorithm:** (1) Let il=1/sqrt(1+λ^2); (2) Generate Y and Z, two Normal(0,1) variates; (3) if $Y>-\xi*il$, return $Y*\lambda*il + Z$; else go to 2.	λ and ξ are real numbers.
Generalized geometric (Francis-Staite and White 2022)[85]	(1) Set *ret* to 1; (2) with probability ρ(*ret*), add 1 to *ret* and repeat this step; otherwise, return *ret*.	$0 \le \rho(k) \le 1$ for each k.
Generalized Sibuya (Kozubowski and Podgórski 2018)[86]	(1) Set *ret* to 1; (2) with probability $\alpha/(\nu+ret)$, return *ret*; otherwise, add 1 to *ret* and repeat this step.	$\alpha < \nu + 1$, and $\nu \ge 0$.[87]
Himanshu (Agarwal and Pandey 2022)[88]	(1) Set *ret* to 0; (2) flip coin that shows heads with probability p, n times; (3) if any flip shows 0 (tails), add 1 to *ret* and go to 2; otherwise, return *ret*.	$0 \le p \le 1$; $n \ge 1$ is an integer.
Tilted beta (Hahn and López Martín 2005)[89]	(1) With probability θ, return a beta(α, β) variate; (2) Generate a uniform variate in (0, 1), call it x; (3) Flip coin that returns 1 with probability x, and another that returns 1 with probability ν; (4) If both coins return 1 or both return 0, return x; otherwise go to step 2.	$0 \le \theta \le 1$; $0 \le \nu \le 1$; $\alpha>0$; $\beta>0$.

7 Random Variate Generation via

Quantiles

This note is about generating random variates from a non-discrete distribution via *inverse transform sampling*, using uniform **partially-sampled random numbers (PSRNs)**.

In this section:

- A distribution's *quantile function* (also known as *inverse cumulative distribution function* or *inverse CDF*) is a nowhere decreasing function that maps uniform random variates greater than 0 and less than 1 to numbers that follow the distribution.
- A *uniform PSRN* is ultimately a number that lies in an interval; it contains a sign, an integer part, and a fractional part made up of digits sampled on demand.

Take the following situation:

- Let $f(.)$ be a function applied to a or b before calculating the quantile.
- Let $Q(z)$ be the quantile function for the desired distribution.
- Let x be a random variate in the form of a uniform PSRN, so that this PSRN will lie in the interval $[a, b]$. If $f(t) = t$ (the identity function), the PSRN x must have a positive sign and an integer part of 0, so that the interval $[a, b]$ is either the interval [0, 1] or a closed interval in [0, 1], depending on the PSRN's fractional part. For example, if the PSRN is 2.147..., then the interval is [2.147, 2.148].
- Let β be the digit base of digits in x's fractional part (such as 2 for binary).
- Suppose $Q(z)$ is continuous on the open interval (a, b).

Then the following algorithm transforms that number to a random variate for the desired distribution, which comes within the desired error tolerance of ε with probability 1 (see (Devroye and Gravel 2020)[90]):

1. Generate additional digits of x uniformly at random—thus shortening the interval $[a, b]$—until a lower bound of $Q(f(a))$ and an upper bound of $Q(f(b))$ differ by no more than $2*\varepsilon$. Call the two bounds low and $high$, respectively.
2. Return $low+(high-low)/2$.

In some cases, it may be possible to calculate the needed digit size in advance.

As one example, if $f(t) = t$ (the identity function) and the quantile function is *Lipschitz continuous* with Lipschitz constant L or less on the interval $[a, b]$[91], then the following algorithm generates a quantile with error tolerance ε:

1. Let d be ceil((ln(max(1,L)) − ln(ε)) / ln(β)). For each digit among the first d digits in x's fractional part, if that digit is unsampled, set it to a digit chosen uniformly at random.
2. The PSRN x now lies in the interval $[a, b]$. Calculate lower and upper bounds of $Q(a)$ and $Q(b)$, respectively, that are within $\varepsilon/2$ of the true quantiles, call the bounds *low* and *high*, respectively.
3. Return $low+(high-low)/2$.

This algorithm chooses a random interval of size equal to β^d, and because the quantile function is Lipschitz continuous, the values at the interval's bounds are guaranteed to vary by no more than $2*\varepsilon$ (actually ε, but the calculation in step 2 adds an additional error of at most ε), which is needed to meet the tolerance ε (see also Devroye and Gravel 2020[92]).

A similar algorithm can exist even if the quantile function Q is not Lipschitz continuous on the interval $[a, b]$.

Specifically, if—

- $f(t) = t$ (the identity function),
- Q on the interval $[a, b]$ is continuous and has a minimum and maximum, and
- Q on $[a, b]$ admits a continuous and strictly increasing function $\omega(\delta)$ as a *modulus of continuity*,

then d in step 1 above can be calculated as—

$$\max(0, \operatorname{ceil}(-\ln(\omega^{-1}(\varepsilon))/\ln(\beta))),$$

where $\omega^{-1}(\varepsilon)$ is the inverse of the modulus of continuity. (Loosely speaking, a modulus of continuity $\omega(\delta)$ gives the quantile function's maximum-minus-minimum in a window of size δ, and the inverse modulus $\omega^{-1}(\varepsilon)$ finds a window small enough that the quantile function differs by no more than ε in the window.[93]).[94]

For example—

- if Q is Lipschitz continuous[95] with Lipschitz constant L or less on $[a, b]$, then the function is no "steeper" than that of $\omega(\delta) = L*\delta$, so $\omega^{-1}(\varepsilon) = \varepsilon/L$, and
- if Q is Hölder continuous with Hölder constant M or less and Hölder exponent α on that interval [96], then the function is no "steeper" than that of $\omega(\delta) = M*\delta^{\alpha}$, so $\omega^{-1}(\varepsilon) = (\varepsilon/M)^{1/\alpha}$.

The algorithms given earlier in this section have a disadvantage: the desired error tolerance has to be made known to the algorithm in advance. (Indeed, for this reason, the algorithms don't satisfy **desirable properties for PSRN samplers**.) To generate a quantile to any error tolerance (even if the tolerance is not known in advance), a rejection sampling approach is needed. For this to work:

- The target distribution must have a probability density function (PDF), as is the case with the normal and exponential distributions.
- That PDF, or a function proportional to it, must be known, must be less than or equal to a finite number, and must be continuous "almost everywhere" (the set of discontinuous points is "zero-volume", that is, has Lebesgue measure zero) (see also (Devroye and Gravel 2020)[97]).

Here is a sketch of how this rejection sampler might work:

1. After using one of the algorithms given earlier in this section to sample digits of x as needed, let a and b be x's upper and lower bounds. Calculate lower and upper bounds of the quantiles of $f(a)$ and $f(b)$ (the bounds are [*alow*, *ahigh*] and [*blow*, *bhigh*] respectively).
2. Given the target function's PDF or a function proportional to it, sample a uniform PSRN, y, in the interval [*alow*, *bhigh*] using an arbitrary-precision rejection sampler such as Oberhoff's method (described in an **appendix to the PSRN article**).
3. Accept y (and return it) if it clearly lies in [*ahigh*, *blow*]. Reject y (and go to the previous step) if it clearly lies outside [*alow*, *bhigh*]. If y clearly lies in [*alow*, *ahigh*] or in [*blow*, *bhigh*], generate more digits of x, uniformly at random, and go to the first step.
4. If y doesn't clearly fall in any of the cases in the previous step, generate more digits of y, uniformly at random, and go to the previous step.

8 Batching Random Samples via Randomness Extraction

Devroye and Gravel (2020)[98] suggest the following randomness extractor to reduce the number of random bits needed to produce a batch of samples by a sampling algorithm. The extractor works based on the probability that the algorithm consumes X random bits given that it produces a specific output Y (or $P(X \mid Y)$ for short):

1. Start with the interval [0, 1].
2. For each pair (X, Y) in the batch, the interval shrinks from below by $P(X-1 \mid Y)$ and from above by $P(X \mid Y)$. (For example, if [0.2, 0.8] (range 0.6) shrinks from below by 0.1 and from above by 0.8, the new interval is [0.2+0.1*0.6, 0.2+0.8*0.6] = [0.26, 0.68]. For correctness, though, the interval is not allowed to shrink to a single point, since otherwise step 3 would run forever.)
3. Extract the bits, starting from the binary point, that the final interval's lower and upper bound have in common (or 0 bits if the upper bound is 1). (For example, if the final interval is [0.101010..., 0.101110...] in binary, the bits 1, 0, 1 are extracted, since the common bits starting from the point are 101.)

After a sampling method produces an output Y, both X (the number of random bits the sampler consumed) and Y (the output) are added to the batch and fed to the extractor, and new bits extracted this way are added to a queue for the sampling method to use to produce future outputs. (Notice that the number of bits extracted by the algorithm above grows as the batch grows, so only the new bits extracted this way are added to the queue this way.)

The issue of finding $P(X \mid Y)$ is now discussed. Generally, if the sampling method implements a random walk on a binary tree that is driven by unbiased random bits and has leaves labeled with one outcome each (Knuth and Yao 1976)[99], $P(X \mid Y)$ is found as follows (and Claude Gravel clarified to me that this is the intention of the extractor algorithm): Take a weighted count of all leaves labeled Y up to depth X (where the weight for depth z is $1/2^z$), then divide it by a weighted count of all leaves labeled Y at all depths (for instance, if the tree has two leaves labeled Y at z=2, three at z=3, and three at z=4, and X is 3, then $P(X \mid Y)$ is $(2/2^2+3/2^3) / (2/2^2+3/2^3+3/2^4)$). In the special case where the tree has at most 1 leaf labeled Y at every

depth, this is implemented by finding $P(Y)$, or the probability to output Y, then chopping $P(Y)$ up to the X^{th} binary digit after the point and dividing by the original $P(Y)$ (for instance, if X is 4 and P(Y) is 0.101011..., then $P(X \mid Y)$ is 0.1010 / 0.101011...).

Unfortunately, $P(X \mid Y)$ is not easy to calculate when the number of values Y can take on is large or even unbounded. In this case, I can suggest the following ad hoc algorithm, which uses a randomness extractor that takes *bits* as input, such as the von Neumann, Peres, or Zhou–Bruck extractor (see "**Notes on Randomness Extraction**"). The algorithm counts the number of bits it consumes (X) to produce an output, then feeds X to the extractor as follows.

1. Let z be abs($X-lastX$), where $lastX$ is either the last value of X fed to this extractor for this batch or 0 if there is no such value.
2. If z is greater than 0, feed the bits of z from most significant to least significant to a queue of extractor inputs.
3. Now, when the sampler consumes a random bit, it checks the input queue. As long as 64 bits or more are in the input queue, the sampler dequeues 64 bits from it, runs the extractor on those bits, and adds the extracted bits to an output queue. (The number 64 can instead be any even number greater than 2.) Then, if the output queue is not empty, the sampler dequeues a bit from that queue and uses that bit; otherwise it generates an unbiased random bit as usual.

9 Sampling Distributions Using Incomplete Information

The Bernoulli factory problem (the problem of turning one biased coin into another biased coin; see "**Bernoulli Factory Algorithms**") is a special case of the problem of **sampling a probability distribution with unknown parameters**. This problem can be described as sampling from a new distribution using an *oracle* (black box) that produces numbers of an incompletely known distribution. In the Bernoulli factory problem, this oracle is a *coin that shows heads or tails where the probability of heads is unknown*. The rest of this section deals with oracles that go beyond coins.

Algorithm 1. Suppose there is an oracle that produces independent random variates on a closed interval $[a, b]$, and these numbers have an unknown mean of μ. The goal is now to produce nonnegative random variates whose expected value ("long-run average") is $f(\mu)$. Unless f is constant, this is possible if and only if—

- f is continuous on the closed interval, and
- $f(\mu)$ is greater than or equal to ε*min($(\mu - a)^n$, $(b - \mu)^n$) for some integer n and some ε greater than 0 (loosely speaking, f is nonnegative and neither touches 0 in the interior of the interval nor moves away from 0 more slowly than a polynomial)

(Jacob and Thiery 2015)[100]. (Here, a and b are both rational numbers and may be less than 0.)

In the algorithm below, let K be a rational number greater than the maximum value of f on the closed interval $[a, b]$, and let $g(\lambda) = f(a + (b-a)*\lambda)/K$.

1. Create a λ input coin that does the following: "Take a number from the oracle, call it x. With probability $(x-a)/(b-a)$ (see note below), return 1. Otherwise, return 0."
2. Run a Bernoulli factory algorithm for $g(\lambda)$, using the λ input coin. Then return K times the result.

> **Note:** The check "With probability $(x-a)/(b-a)$" is exact if the oracle produces only rational numbers. Otherwise, calculating the probability can lead to numerical errors unless care is taken (see note 2 in "Distributions with nowhere increasing or nowhere decreasing weights", above). With uniform partially-sampled random numbers (PSRNs), the check can be implemented as follows. Let x be a uniform PSRN representing a number generated by the oracle. Set y to **RandUniformFromReal**$(b-a)$, then the check succeeds if **RandLess**$(y,$ **UniformAddRational**$(x, -a))$ returns 1, and fails otherwise.
>
> **Example:** Suppose an oracle produces random variates in the interval [3, 13] with unknown mean μ, and the goal is to use the oracle to produce nonnegative random variates with mean $f(\mu) = -319/100 + \mu*103/50 - \mu^2*11/100$, which is a polynomial with Bernstein coefficients [2, 9, 5] in the given interval. Then since 8 is greater than the maximum of f in that interval, $g(\lambda)$ is a

> degree-2 polynomial in the interval [0, 1] that has Bernstein coefficients [2/8, 9/8, 5/8]. *g* can't be simulated as is, though, but increasing *g*'s degree to 3 leads to the Bernstein coefficients [1/4, 5/6, 23/24, 5/8], which are all less than 1 so that the following algorithm can be used (see "**Certain Polynomials**"):
>
> 1. Set *heads* to 0.
> 2. Generate three random variates from the oracle (which must produce random variates in the interval [3, 13]). For each number x: With probability $(x-3)/(10-3)$, add 1 to *heads*.
> 3. Depending on *heads*, return 8 (that is, 1 times the upper bound) with the given probability, or 0 otherwise: *heads*=0 → probability 1/4; 1 → 5/6; 2 → 23/24; 3 → 5/8.

Algorithm 2. Say there is an oracle in the form of a fair die. The number of faces of the die, n, is at least 2 but otherwise unknown. Each face shows a different integer 0 or greater and less than n. The question arises: Which probability distributions based on the number of faces can be sampled with this oracle? This question was studied in the French-language dissertation of R. Duvignau (2015, section 5.2)[101], and the following are four of these distributions.

Bernoulli 1/n. It's trivial to generate a Bernoulli variate that is 1 with probability $1/n$ and 0 otherwise: just take a number from the oracle and return either 1 if that number is 0, or 0 otherwise. Alternatively, take two numbers from the oracle and return either 1 if both are the same, or 0 otherwise (Duvignau 2015, p. 153)[102].

Random variate with mean n. Likewise, it's trivial to generate variates with a mean of n: Do "Bernoulli 1/n" trials as described above until a trial returns 0, then return the number of trials done this way. (This is related to the ambiguously defined "geometric" random variates.)

Binomial with parameters n and 1/n. Using the oracle, the following algorithm generates a binomial variate of this kind (Duvignau 2015, Algorithm 20)[103]:

1. Take items from the oracle until the same item is taken twice.
2. Create a list consisting of the items taken in step 1, except for the last item taken, then shuffle that list.
3. In the shuffled list, count the number of items that didn't change position after being shuffled, then return that number.

Binomial with parameters n and k/n. Duvignau 2015 also includes an algorithm (Algorithm 25) to generate a binomial variate of this kind using the oracle (where k is a known integer such that $0 < k$ and $k \leq n$):

1. Take items from the oracle until k different items were taken this way. Let U be a list of these k items, in the order in which they were first taken.
2. Create an empty list L.
3. For each integer i satisfying $0 \leq i < k$:
    1. Create an empty list M.
    2. Take an item from the oracle. If the item is in U at a position **less than *i*** (positions start at 0), repeat this substep. Otherwise, if the item is not in M, add it to M and repeat this substep. Otherwise, go to the next substep.
    3. Shuffle the list M, then add to L each item that didn't change position after being shuffled (if not already present in L).
4. For each integer i satisfying $0 \leq i < k$:
    1. Let P be the item at position i in U.
    2. Take an item from the oracle. If the item is in U at position ***i* or less** (positions start at 0), repeat this substep.
    3. If the last item taken in the previous substep is in U at a position **greater than *i***, add P to L (if not already present).
5. Return the number of items in L.

> **Note:** Duvignau proved a result (Theorem 5.2) that answers the question: Which probability distributions based on the unknown n can be sampled with the oracle?[104] The result applies to a family of (discrete) distributions with the same unknown parameter n, starting with either 1 or a greater integer. Let Supp(m) be the set of values taken on by the distribution with parameter equal to m. Then that family can be sampled using the oracle (with or without additional randomness) if and only if:
>
> - There is a computable function $f(k)$ that outputs a positive number.
> - For each n, Supp(n) is included in Supp(n+1).
> - For every k and for every $n \geq 2$ starting with the first n for which k is in Supp(n), the probability of seeing k given parameter n is at least $(1/n)^{f(k)}$.

> Moreover, by Proposition 5.5 of Duvignau, a family meeting the conditions above can be sampled without additional randomness (besides the oracle) if and only if Supp(1) has no more than one element.
>
> **Example:** Let $n \geq 2$ be an integer.
> The family of Bernoulli distributions, taking on 1 with probability $\exp(-n)$ and 0 otherwise, cannot be simulated this way, because that probability decays faster than the rate $(1/n)^{f(1)}$ for any f.
> This is consistent with the results for *Bernoulli factories* (Keane and O'Brien 1994)[105], where a coin that shows heads with unknown probability $\lambda = 1/n$ cannot be turned into a coin that shows heads with probability $g(\lambda) = \exp(-1/\lambda) = \exp(-n)$ since g is not polynomially bounded (away from 0).
> However, a Bernoulli family, taking on 1 with probability $h(n) = (1+\ln(n))/n$ and 0 with probability $1-h(n)$, *can* be simulated, because $\min(h(n), 1-h(n)) \geq (1/n)^3$.

9.1 Additional Algorithms

The following algorithms are included here because they require applying an arbitrary function (such as $f(\lambda)$) to a potentially irrational number.

Algorithm 3. Suppose there is an *oracle* that produces independent random real numbers whose expected value ("long-run average") is a known or unknown mean. The goal is now to produce nonnegative random variates whose expected value is the mean of $f(X)$, where X is a number produced by the oracle. This is possible whenever—

- f has a finite lower bound and a finite upper bound on its domain, and
- the mean of $f(X)$ is not less than δ, where δ is a known rational number greater than 0.

The algorithm to achieve this goal follows (see Lee et al. 2014[106]):

1. Let m be a rational number equal to or greater than the maximum value of $\operatorname{abs}(f(\mu))$ anywhere. Create a ν input coin that does the following: "Take a number from the oracle, call it x. With probability $\operatorname{abs}(f(x))/m$, return a number that is 1 if $f(x) < 0$ and 0 otherwise. Otherwise, repeat this process."

2. Use one of the **linear Bernoulli factories** to simulate 2*ν (2 times the ν coin's probability of heads), using the ν input coin, with $\epsilon = \delta/m$. If the factory returns 1, return 0. Otherwise, take a number from the oracle, call it ξ, and return abs($f(\xi)$).

> **Example:** An example from Lee et al. (2014)[107]. Say the oracle produces uniform random variates in [0, 3*π], and let $f(\nu) = \sin(\nu)$. Then the mean of $f(X)$ is 2/(3*π), which is greater than 0 and found in SymPy by `sympy.stats.E(sin(sympy.stats.Uniform('U',0,3*pi)))`, so the algorithm can produce nonnegative random variates whose expected value ("long-run average") is that mean.
>
> **Notes:**
>
> 1. Averaging to the mean of $f(X)$ (that is, $\mathbf{E}[f(X)]$ where $\mathbf{E}[.]$ means expected value or "long-run average") is not the same as averaging to $f(\mu)$ where μ is the mean of the oracle's numbers (that is, $f(\mathbf{E}[X])$). For example, if X is 0 or 1 with equal probability, and $f(\nu) = \exp(-\nu)$, then $\mathbf{E}[f(X)] = \exp(0) + (\exp(-1) - \exp(0))*(1/2)$, and $f(\mathbf{E}[X]) = f(1/2) = \exp(-1/2)$.
>
> 2. (Lee et al. 2014, Corollary 4)[108]: If $f(\mu)$ is known to return only values in the interval [a, c], the mean of $f(X)$ is not less than δ, $\delta > b$, and δ and b are known numbers, then Algorithm 2 can be modified as follows:
>
>   - Use $f(\nu) = f(\nu) - b$, and use $\delta = \delta - b$.
>   - m is taken as max($b-a$, $c-b$).
>   - When Algorithm 2 finishes, add b to its return value.
>
> 3. The check "With probability abs($f(x)$)/m" is exact if the oracle produces only rational numbers *and* if $f(x)$ outputs only rational numbers. If the oracle or f can produce irrational numbers (such as numbers that follow a beta distribution or another non-discrete distribution), then calculating the probability can lead to numerical errors unless care is taken (see note 2 in "Distributions with nowhere increasing or nowhere decreasing weights", above).

Algorithm 4. Suppose there is an *oracle* that produces independent random real numbers that are all greater than or equal to a (which is a known rational number), whose mean (μ) is unknown. The goal is to use the oracle to produce nonnegative random variates with mean

$f(\mu)$. This is possible only if f is 0 or greater everywhere in the interval $[a, \infty)$ and is nowhere decreasing in that interval (Jacob and Thiery 2015)[109]. This can be done using the algorithm below. In the algorithm:

- $f(\mu)$ must be a function that can be written as—
  $c[0]*z^0 + c[1]*z^1 + ...$,
  which is an infinite series where $z = \mu - a$ and all $c[i]$ are 0 or greater.
- ψ is a rational number close to 1, such as 95/100. (The exact choice is arbitrary and can be less or greater for efficiency purposes, but must be greater than 0 and less than 1.)

The algorithm follows.

1. Set *ret* to 0, *prod* to 1, k to 0, and w to 1. (w is the probability of taking k or more numbers from the oracle in a single run of the algorithm.)
2. If k is greater than 0: Take a number from the oracle, call it x, and multiply *prod* by $x - a$.
3. Add $c[k]*prod/w$ to *ret*.
4. Multiply w by ψ and add 1 to k.
5. With probability ψ, go to step 2. Otherwise, return *ret*.

Now, assume the oracle's numbers are all less than or equal to b (rather than greater than or equal to a), where b is a known rational number. Then f must be 0 or greater everywhere in $(-\infty, b]$ and be nowhere increasing there (Jacob and Thiery 2015)[110], and the algorithm above can be used with the following modifications: (1) In the note on the infinite series, $z = b - \mu$; (2) in step 2, multiply *prod* by $b - x$ rather than $x - a$.

> **Note:** This algorithm is exact if the oracle produces only rational numbers *and* if all $c[i]$ are rational numbers. Otherwise, the algorithm can introduce numerical errors unless care is taken (see note 2 in "Distributions with nowhere increasing or nowhere decreasing weights", above). See also note 3 on the previous algorithm.

10 Acknowledgments

Due to a suggestion by Michael Shoemate who suggested it was "easy to get lost" in this and related articles, some sections that related to geometric distributions were moved here. He also noticed a minor error which was corrected.

11 Notes

12 License

---

1. K. Bringmann, F. Kuhn, et al., “Internal DLA: Efficient Simulation of a Physical Growth Model.” In: *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*, 2014.↩

2. "x is odd" means that x is an integer and not divisible by 2. This is true if x − 2*floor(x/2) equals 1, or if x is an integer and the least significant bit of abs(x) is 1.↩

3. choose(n, k) = (1*2*3*...*n)/((1*...*k)*(1*...*(n−k))) = n!/(k! * (n − k)!) is a *binomial coefficient*, or the number of ways to choose k out of n labeled items. It can be calculated, for example, by calculating i/(n−i+1) for each integer i satisfying n−k+1 ≤ i ≤ n, then multiplying the results (Yannis Manolopoulos. 2002. "**Binomial coefficient computation: recursion or iteration?**", SIGCSE Bull. 34, 4 (December 2002), 65–67). Note that for every m>0, choose(m, 0) = choose(m, m) = 1 and choose(m, 1) = choose(m, m−1) = m; also, in this document, choose(n, k) is 0 when k is less than 0 or greater than n.↩

4. Devroye, L., ***Non-Uniform Random Variate Generation***, 1986.↩

5. R. Schumacher, "**Rapidly Convergent Summation Formulas involving Stirling Series**", arXiv:1602.00336v1 [math.NT], 2016.↩

6. "x is odd" means that x is an integer and not divisible by 2. This is true if $x - 2*\text{floor}(x/2)$ equals 1, or if x is an integer and the least significant bit of abs(x) is 1.↩

7. Farach-Colton, M. and Tsai, M.T., 2015. Exact sublinear binomial sampling. *Algorithmica* 73(4), pp. 637-651.↩

8. Bringmann, K., and Friedrich, T., 2013, July. Exact and efficient generation of geometric random variates and random graphs, in *International Colloquium on Automata, Languages, and Programming* (pp. 267-278).↩

9. "x is even" means that x is an integer and divisible by 2. This is true if $x - 2*\text{floor}(x/2)$ equals 0, or if x is an integer and the least significant bit of abs(x) is 0.↩

10. "x is even" means that x is an integer and divisible by 2. This is true if $x - 2*\text{floor}(x/2)$ equals 0, or if x is an integer and the least significant bit of abs(x) is 0.↩

11. Ghosh, A., Roughgarden, T., and Sundararajan, M., "Universally Utility-Maximizing Privacy Mechanisms", *SIAM Journal on Computing* 41(6), 2012.↩

12. Li, L., 2021. Bayesian Inference on Ratios Subject to Differentially Private Noise (Doctoral dissertation, Duke University).↩

13. Canonne, C., Kamath, G., Steinke, T., "**The Discrete Gaussian for Differential Privacy**", arXiv:2004.00010 [cs.DS], 2020.↩

14. Chewi, Sinho, Patrik R. Gerber, Chen Lu, Thibaut Le Gouic, and Philippe Rigollet. "**Rejection sampling from shape-constrained distributions in sublinear time**." In International Conference on Artificial Intelligence and Statistics, pp. 2249-2265. PMLR, 2022.↩

15. Hans-J. Boehm. 1987. Constructive Real Interpretation of Numerical Programs. In Proceedings of the SIGPLAN '87 Symposium on Interpreters and Interpretive Techniques. 214-221↩

16. Boehm, Hans-J. "Towards an API for the real numbers." In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 562-576. 2020.↩

17. Chewi, Sinho, Patrik R. Gerber, Chen Lu, Thibaut Le Gouic, and Philippe Rigollet. "**Rejection sampling from shape-constrained distributions in sublinear time**." In International Conference on Artificial Intelligence and Statistics, pp. 2249-2265. PMLR, 2022.↩

18. Huijben, I.A., Kool, W., Paulus, M.B. and Van Sloun, R.J., 2022. A Review of the Gumbel-max Trick and its Extensions for Discrete Stochasticity in Machine Learning. IEEE Transactions on Pattern Analysis and Machine Intelligence. Also in **https://arxiv.org/pdf/2110.01515**↩

19. Or as $-\ln(E)$, where E is an exponential random variate with rate 1.↩

20. Huijben, I.A., Kool, W., Paulus, M.B. and Van Sloun, R.J., 2022. A Review of the Gumbel-max Trick and its Extensions for Discrete Stochasticity in Machine Learning. IEEE Transactions on Pattern Analysis and Machine Intelligence. Also in **https://arxiv.org/pdf/2110.01515**↩

21. Tillé, Y., "Remarks on some misconceptions about unequal probability sampling without replacement", Computer Science Review 47 (Feb. 2023).↩

22. Chakraborty, A., Vardeman, S. B., Modeling and inference for mixtures of simple symmetric exponential families of p-dimensional distributions for vectors with binary coordinates, Stat Anal Data Min: The ASA Data Sci Journal. 2021; 14: 352– 365. **https://doi.org/10.1002/sam.11528**↩

23. Devroye, L., ***Non-Uniform Random Variate Generation***, 1986.↩

24. Ahmad, Z. et al. "Recent Developments in Distribution Theory: A Brief Survey and Some New Generalized Classes of distributions." Pakistan Journal of Statistics and Operation Research 15 (2019): 87-110.↩

25. Jones, M. C. "On families of distributions with shape parameters." International Statistical Review 83, no. 2 (2015): 175-192.↩

26. Eugene, N., Lee, C., Famoye, F., "Beta-normal distribution and its applications", *Commun. Stat. Theory Methods* 31, 2002.↩

27. Barreto-Souza, Wagner and Alexandre B. Simas. "The exp-G family of probability distributions." *Brazilian Journal of Probability and Statistics* 27, 2013. Also in arXiv:1003.1727v1 [stat.ME], 2010.↩

28. Mahdavi, Abbas, and Debasis Kundu. "A new method for generating distributions with an application to exponential distribution." *Communications in Statistics -- Theory and Methods* 46, no. 13 (2017): 6543-6557.↩

29. M. C. Jones. Letter to the Editor concerning “A new method for generating distributions with an application to exponential distribution” and “Alpha power Weibull distribution: Properties and applications”, *Communications in Statistics - Theory and Methods* 47 (2018).↩

30. Lehmann, E.L., "The power of rank tests", Annals of Mathematical Statistics 24(1), March 1953.↩

31. Durrans, S.R., "Distributions of fractional order statistics in hydrology", Water Resources Research 28 (1992).↩

32. Mudholkar, G. S., Srivastava, D. K., "Exponentiated Weibull family for analyzing bathtub failure-rate data", *IEEE Transactions on Reliability* 42(2), 299-302, 1993.↩

33. Shaw, W.T., Buckley, I.R.C., "The alchemy of probability distributions: Beyond Gram-Charlier expansions, and a skew-kurtotic-normal distribution from a rank transmutation map", 2007.↩

34. Devroye, L., ***Non-Uniform Random Variate Generation***, 1986.↩

35. Granzotto, D.C.T., Louzada, F., et al., "Cubic rank transmuted distributions: inferential issues and applications", *Journal of Statistical Computation and Simulation*, 2017.↩

36. Granzotto, D.C.T., Louzada, F., et al., "Cubic rank transmuted distributions: inferential issues and applications", *Journal of Statistical Computation and Simulation*, 2017.↩

37. Grassia, A., "On a family of distributions with argument between 0 and 1 obtained by transformation of the gamma and derived compound distributions", *Australian Journal of Statistics*, 1977.↩

38. Alzaatreh, A., Famoye, F., Lee, C., "A new method for generating families of continuous distributions", *Metron* 71:63–79 (2013).↩

39. Aljarrah, M.A., Lee, C. and Famoye, F., "On generating T-X family of distributions using quantile functions", Journal of Statistical Distributions and Applications,1(2), 2014.↩

40. Gleaton, J.U., Lynch, J. D., "Properties of generalized log-logistic families of lifetime distributions", *Journal of Probability and Statistical Science* 4(1), 2006.↩

41. Hosseini, B., Afshari, M., "The Generalized Odd Gamma-G Family of Distributions: Properties and Applications", *Austrian Journal of Statistics* vol. 47, Feb. 2018.↩

42. N.H. Al Noor and N.K. Assi, "Rayleigh-Rayleigh Distribution: Properties and Applications", *Journal of Physics: Conference Series* 1591, 012038 (2020). The underlying Rayleigh distribution uses a parameter θ (or λ), which is different from *Mathematica*'s parameterization with σ = sqrt($1/\theta^2$) = sqrt($1/\lambda^2$). The first Rayleigh distribution uses θ and the second, λ.↩

43. Boshi, M.A.A., et al., "Generalized Gamma – Generalized Gompertz Distribution", *Journal of Physics: Conference Series* 1591, 012043 (2020).↩

44. Tahir, M.H., Cordeiro, G.M., "Compounding of distributions: a survey and new generalized classes", *Journal of Statistical Distributions and Applications* 3(13), 2016.↩

45. Pérez-Casany, M., Valero, J., and Ginebra, J. (2016). Random-Stopped Extreme distributions. International Conference on Statistical Distributions and Applications.↩

46. Amponsah, C.K., Kozubowski, T.J. & Panorska, A.K. A general stochastic model for bivariate episodes driven by a gamma sequence. J Stat Distrib App 8, 7 (2021). **https://doi.org/10.1186/s40488-021-00120-5**↩

47. Johnson, N. L., Kemp, A. W., and Kotz, S. (2005). Univariate discrete distributions.↩

48. Duarte-López, A., Pérez-Casany, M. and Valero, J., 2021. Randomly stopped extreme Zipf extensions. Extremes, pp.1-34.↩

49. This is simplified from the paper because Y can take on only values greater than 0 so that the probability of getting 0 is 0.↩

50. Akdoğan, Y., Kus, C., et al., "Geometric-Zero Truncated Poisson Distribution: Properties and Applications", *Gazi University Journal of Science* 32(4), 2019.↩

51. Amponsah, C.K., Kozubowski, T.J. & Panorska, A.K. A general stochastic model for bivariate episodes driven by a gamma sequence. J Stat Distrib App 8, 7 (2021). **https://doi.org/10.1186/s40488-021-00120-5**↩

52. Barreto-Souza, W.: "Bivariate gamma-geometric law and its induced Lévy process", Journal of Multivariate Analysis 109 (2012).↩

53. Kuş, C., "A new lifetime distribution", *Computational Statistics & Data Analysis* 51 (2007).↩

54. Cancho, Vicente G., Franscisco Louzada-Neto, and Gladys DC Barriga. "The Poisson-exponential lifetime distribution." Computational Statistics & Data Analysis 55, no. 1 (2011): 677-686.↩

55. Jodrá, P., "A note on the right truncated Weibull distribution and the minimum of power function distributions", 2020.↩

56. Barreto-Souza, W.: "Bivariate gamma-geometric law and its induced Lévy process", Journal of Multivariate Analysis 109 (2012).↩

57. Keller, A.Z., Kamath A.R., "Reliability analysis of CNC machine tools", *Reliability Engineering* 3 (1982).↩

58. Devroye, L., ***Non-Uniform Random Variate Generation***, 1986.↩

59. Rao, C.R., "Weighted distributions arising out of methods of ascertainment", 1985.↩

60. Ospina, R., Ferrari, S.L.P., "Inflated Beta Distributions", 2010.↩

61. Chakraborty, S., Bhattacharjee, S., "**Modeling proportion of success in high school leaving examination- A comparative study of Inflated Unit Lindley and Inflated Beta distribution**", arXiv:2103.08916 [stat.ME], 2021.↩

62. Mullahy, J., "Specification and testing of some modified count data models", 1986.↩

63. Grassia, A., "On a family of distributions with argument between 0 and 1 obtained by transformation of the gamma and derived compound distributions", *Australian Journal of Statistics*, 1977.↩

64. Akdoğan, Y., Kus, C., et al., "Geometric-Zero Truncated Poisson Distribution: Properties and Applications", *Gazi University Journal of Science* 32(4), 2019.↩

65. Elgohari, Hanaa, and Haitham Yousof. "New Extension of Weibull Distribution: Copula, Mathematical Properties and Data Modeling." Stat., Optim. Inf. Comput., Vol.8, December 2020.↩

66. Marshall, A.W. and Olkin, I., 1997. A new method for adding a parameter to a family of distributions with application to the exponential and Weibull families. Biometrika, 84(3), pp.641-652.↩

67. Rady, E.H.A., Hassanein, W.A., Elhaddad, T.A., "The power Lomax distribution with an application to bladder cancer data", (2016).↩

68. Castellares, F., Lemonte, A.J., Moreno, G., "On the two-parameter Bell-Touchard discrete distribution", _Communications in Statistics
    - Theory and Methods_ 4, (2020).
    ↩

69. The similar Bell–Touchard process is the sum of the first N variates from an infinite sequence of zero-truncated Poisson(a) random variates, where N is the number of events of a Poisson process with rate b*exp(a)−b (Freud, T., Rodriguez, P.M., "**The Bell-Touchard counting process**", arXiv:2203.16737v2 [math.PR], 2022).↩

70. Castellares, F., Lemonte, A.J., Moreno, G., "On the two-parameter Bell-Touchard discrete distribution", _Communications in Statistics
    - Theory and Methods_ 4, (2020).
    ↩

71. Buddana, Amrutha, and Tomasz J. Kozubowski. "Discrete Pareto distributions." Economic Quality Control 29, no. 2 (2014): 143-156.↩

72. Batsidis, A., Lemonte, A.J., "On Goodness-of-Fit Tests for the Neyman Type A Distribution", REVSTAT-Statistical Journal (accepted Nov. 2021).↩

73. Kudryavtsev, A.A., "On the representation of gamma-exponential and generalized negative binomial distributions", Inform. Appl. 13 (2019)↩

74. Saha, M., et al., "**The extended xgamma distribution**", arXiv:1909.01103 [math.ST], 2019.↩

75. McNeil, et al., "Quantitative risk management", 2010.↩

76. Azzalini, A., Capitanio, A., "Distributions generated by perturbation of symmetry with emphasis on a multivariate skew t-distribution." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 65, no. 2 (2003): 367-389.↩

77. Azzalini, A., "An overview on the progeny of the skew-normal family— A personal perspective", Journal of Multivariate Analysis 188, March 2022.↩

78. Azzalini, Adelchi. "A class of distributions which includes the normal ones." Scandinavian journal of statistics (1985): 171-178.↩

79. Gómez-Déniz, Emilio, and E. Calderín-Ojeda. "**On the usefulness of the logarithmic skew normal distribution for describing claims size data**." Mathematical Problems in Engineering 2020 (2020). Lin and Stoyanov (2009, "The logarithmic skew-normal distributions are moment-indeterminate", *Journal of Applied Probability* 46) studied the logarithmic skew normal distribution with μ=0 and σ=1.↩

80. Hahn, Eugene D. "The Tilted Beta-Binomial Distribution in Overdispersed Data: Maximum Likelihood and Bayesian Estimation." Journal of Statistical Theory and Practice 16, no. 3 (2022): 1-22.↩

81. Rubio, F.J. and Steel, M.F.J. (2020), The family of two-piece distributions. Significance, 17: 12-13. **https://doi.org/10.1111/j.1740-9713.2020.01352.x**↩

82. A. Tesei and C. S. Regazzoni, “The asymmetric generalized Gaussian function: a new HOS-based model for generic noise PDFs,” in Proceedings of 8th Workshop on Statistical Signal and

Array Processing, Corfu, Greece, Jun. 1996, pp. 210-213↩

83. Sadeghi, Parastoo, and Mehdi Korki. "Offset-Symmetric Gaussians for Differential Privacy." arXiv preprint arXiv:2110.06412 (2021).↩

84. Henze, Norbert. "A probabilistic representation of the 'skew-normal' distribution." Scandinavian journal of statistics (1986): 271-275. SNE(λ,0) is distributed as Azzalini's skew normal distribution.↩

85. Francis-Staite, Kelli, and Langford White. "**Analysis of sojourn time distributions for semi-Markov models**", arXiv:2206.10865 (2022).↩

86. Kozubowski, Tomasz J., and Krzysztof Podgórski. "A generalized Sibuya distribution." Annals of the Institute of Statistical Mathematics 70, no. 4 (2018): 855-887.↩

87. If $\nu = 0$, this is the ordinary Sibuya distribution.↩

88. Agarwal, A., Pandey, H., "Himanshu distribution and its applications", Bulletin of Mathematics and Statistics Research 10(4), 2022.↩

89. Hahn, E.D., López Martín, M.d.M., "Robust project management with the tilted beta distribution", 2015.↩

90. Devroye, L., Gravel, C., "**Random variate generation using only finitely many unbiased, independently and identically distributed random bits**", arXiv:1502.02539v6 [cs.IT], 2020.↩

91. A Lipschitz continuous function, with Lipschitz constant L, is a continuous function such that $f(x)$ and $f(y)$ are no more than $L*\varepsilon$ apart whenever x and y are points in the domain that are no more than ε apart. Roughly speaking, the function's slope is no "steeper" than that of $L*x$.↩

92. Devroye, L., Gravel, C., "**Random variate generation using only finitely many unbiased, independently and identically distributed random bits**", arXiv:1502.02539v6 [cs.IT], 2020.↩

93. Ker-I Ko makes heavy use of the inverse modulus of continuity in his complexity theory, for example, "Computational complexity of roots of real functions." In *30th Annual Symposium on Foundations of Computer Science*, pp. 204-209. IEEE Computer Society, 1989.↩

94. Here is a sketch of the proof: Because the quantile function $Q(x)$ is continuous on a closed interval, it's uniformly continuous there. For this reason, there is a positive function $\omega^{-1}(\varepsilon)$ such that $Q(x)$ is less than ε-away from $Q(y)$, for every $\varepsilon>0$, whenever x and y lie in that interval and whenever x is less than $\omega^{-1}(\varepsilon)$-away from y. The inverse modulus of continuity is one such function, which is formed by inverting a modulus of continuity admitted by Q, as long as that modulus is continuous and strictly increasing on that interval to make that modulus invertible. Finally, $\max(0, \text{ceil}(-\ln(z)/\ln(\beta)))$ is an upper bound on the number of base-β fractional digits needed to store $1/z$ with an error of at most ε.↩

95. A Lipschitz continuous function, with Lipschitz constant L, is a continuous function such that $f(x)$ and $f(y)$ are no more than $L*\varepsilon$ apart whenever x and y are points in the domain that are no more than ε apart. Roughly speaking, the function's slope is no "steeper" than that of $L*x$.↩

96. A ***Hölder continuous*** function (with M being the *Hölder constant* and α being the *Hölder exponent*) is a continuous function f such that $f(x)$ and $f(y)$ are no more than $M*\delta^{\alpha}$ apart whenever x and y are in the function's domain and no more than δ apart.
Here, α satisfies $0 < \alpha \leq 1$.
Roughly speaking, the function's "steepness" is no greater than that of $M*x^{\alpha}$.↩

97. Devroye, L., Gravel, C., "**Random variate generation using only finitely many unbiased, independently and identically distributed random bits**", arXiv:1502.02539v6 [cs.IT], 2020.↩

98. Devroye, L., Gravel, C., "**Random variate generation using only finitely many unbiased, independently and identically distributed random bits**", arXiv:1502.02539v6 [cs.IT], 2020.↩

99. Knuth, Donald E. and Andrew Chi-Chih Yao. "The complexity of nonuniform random number generation", in *Algorithms and Complexity: New Directions and Recent Results*, 1976.↩

100. Jacob, P.E., Thiery, A.H., "On nonnegative unbiased estimators", Ann. Statist., Volume 43, Number 2 (2015), 769-784.↩

101. Duvignau, R., 2015. Maintenance et simulation de graphes aléatoires dynamiques (Doctoral dissertation, Université de Bordeaux).↩

102. Duvignau, R., 2015. Maintenance et simulation de graphes aléatoires dynamiques (Doctoral dissertation, Université de Bordeaux).↩

103. Duvignau, R., 2015. Maintenance et simulation de graphes aléatoires dynamiques (Doctoral dissertation, Université de Bordeaux).↩

104. There are many distributions that can be sampled using the oracle, by first generating unbiased random bits via randomness extraction methods, but then these distributions won't use the unknown number of faces in general. Duvignau proved Theorem 5.2 for an oracle that outputs *arbitrary* but still distinct items, as opposed to integers, but this case can be reduced to the integer case (see section 4.1.3).↩

105. Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.↩

106. Lee, A., Doucet, A. and Łatuszyński, K., 2014. "**Perfect simulation using atomic regeneration with application to Sequential Monte Carlo**", arXiv:1407.5770v1 [stat.CO].↩

107. Lee, A., Doucet, A. and Łatuszyński, K., 2014. "**Perfect simulation using atomic regeneration with application to Sequential Monte Carlo**", arXiv:1407.5770v1 [stat.CO].↩

108. Lee, A., Doucet, A. and Łatuszyński, K., 2014. "**Perfect simulation using atomic regeneration with application to Sequential Monte Carlo**", arXiv:1407.5770v1 [stat.CO].↩

109. Jacob, P.E., Thiery, A.H., "On nonnegative unbiased estimators", Ann. Statist., Volume 43, Number 2 (2015), 769-784.↩

110. Jacob, P.E., Thiery, A.H., "On nonnegative unbiased estimators", Ann. Statist., Volume 43, Number 2 (2015), 769-784.↩