Miscellaneous Observations on Randomization

Peter Occil

Miscellaneous Observations on Randomization

This version of the document is dated 2023-06-14.

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

[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.

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

[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 $n - k+1 \le i \le n$, 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.

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 = 2floor((1+ β (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, β (n) = ceil(ln(n+1)/ln(2)), or alternatively the minimum number of bits needed to store n (with β (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 $0 < p \leq 1$.

> **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 algo-

[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 $x - 2$*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).

rithm 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 $\leq py$, add 1 to k and multiply pn by 2. (Equivalent to finding the largest $k \geq 0$ such that p*2$k \leq 1$. 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/(2k+2) and less than or equal to 1/(2k) will suffice, as the Bringmann paper points out.)
3. With probability (1 − px/py)2k, 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, 2k), call it m, then with probability (1 − px/py)m, return d*2k+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*$px/py \leq 1$. 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 $r \leq$ floor(($pnum$*i)/$pden$) − 2, return 1. If $r \geq$ 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, 2k) with probability (1 − px/py)m, or − 1 with the opposite probability. It assumes that $2^k(px/py) \leq 1$.] = exp(0) + (exp(− 1) − exp(0))*(1/2), and $f(\mathbf{E}[X]) = f(1/2) = \exp(-1/2)$.

[9]"x is even" means that x is an integer and divisible by 2. This is true if x − 2*floor(x/2) equals 0, or if x is an integer and the least significant bit of abs(x) is 0.

> 2. (Lee et al. 2014, Corollary 4)[10]: 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 $\text{abs}(f(x))/m$" 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)[11]. This can be done using the algorithm below. In the algorithm:

- $f(\ \mu\ )$ must be a function that can be written as—$c[0]*z0\ +\ c[1]*z1\ +$ ...,which is an infinite series where $z = \mu - a$ and all $c[i]$ are 0 or greater.
-   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   and add 1 to k.
5. With probability , 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)[12], 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 $x - a$.

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

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

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

> **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.

4 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.

5 Notes

6 License