More Algorithms for Arbitrary-Precision Sampling

This version of the document is dated 2020-10-06.

Peter Occil

This page contains additional algorithms for arbitrary-precision sampling of continuous distributions, Bernoulli factory algorithms (biased-coin to biased-coin algorithms), and algorithms to simulate irrational probabilities. These samplers are designed to not rely on floating-point arithmetic. They may depend on algorithms given in the following pages:

- **Partially-Sampled Random Numbers for Accurate Sampling of the Beta, Exponential, and Other Continuous Distributions**
- **Bernoulli Factory Algorithms**

1 Bernoulli Factories and Irrational Probability Simulation

1.1 Certain Numbers Based on the Golden Ratio

The following algorithm given by Fishman and Miller (2013)**[1]** finds the continued fraction expansion of certain numbers described as—

- $G(m, \ell) = (m + \text{sqrt}(m^2 + 4 * \ell))/2$
  or $(m - \text{sqrt}(m^2 + 4 * \ell))/2$,

whichever results in a real number greater than 1, where m is a positive integer and ℓ is either 1 or −1. In this case, $G(1, 1)$ is the golden ratio.

First, define the following operations:

- **Get the previous and next Fibonacci-based number given *k*, *m*, and ℓ**:
  1. If k is 0 or less, return an error.
  2. Set $g0$ to 0, $g1$ to 1, x to 0, and y to 0.
  3. Do the following k times: Set y to $m * g1 + \ell * g0$, then set x to $g0$, then set $g0$ to $g1$, then set $g1$ to y.
  4. Return x and y, in that order.
- **Get the partial denominator given *pos*, *k*, *m*, and ℓ** (this partial denominator is part of the continued fraction expansion found by Fishman and Miller):
  1. **Get the previous and next Fibonacci-based number given *k*, *m*, and ℓ**, call them p and n, respectively.
  2. If ℓ is 1 and k is odd, return $p + n$.
  3. If ℓ is −1 and pos is 0, return $n - p - 1$.
  4. If ℓ is 1 and pos is 0, return $(n + p) - 1$.
  5. If ℓ is −1 and pos is even, return $n - p - 2$. (The paper had an error here; the correction given here was verified by Miller via personal communication.)

6. If ℓ is 1 and *pos* is even, return $(n + p) - 2$.
7. Return 1.

An application of the continued fraction algorithm is the following algorithm that generates 1 with probability $G(m, \ell)^{-k}$ and 0 otherwise, where k is an integer that is 1 or greater (see "Continued Fractions" in my page on Bernoulli factory algorithms). The algorithm starts with $pos = 0$, then the following steps are taken:

1. **Get the partial denominator given *pos*, *k*, *m*, and ℓ**, call it *kp*.
2. With probability $kp/(1 + kp)$, return a number that is 1 with probability $1/kp$ and 0 otherwise.
3. Run this algorithm recursively, but with $pos = pos + 1$. If the algorithm returns 1, return 0. Otherwise, go to step 2.

2 Arbitrary-Precision Samplers

2.1 Rayleigh Distribution

The following is an arbitrary-precision sampler for the Rayleigh distribution with parameter *s*, which is a rational number greater than 0.

1. Set *k* to 0, and set *y* to $2 * s * s$.
2. With probability $\exp(-(k * 2 + 1)/y)$, go to step 3. Otherwise, add 1 to *k* and repeat this step. (The probability check should be done with the **exp(−x/y) algorithm** in "**Bernoulli Factory Algorithms**", with $x/y = (k * 2 + 1)/y$.)
3. (Now we sample the piece located at $[k, k + 1)$.) Create a positive-sign zero-integer-part uniform PSRN, and create an input coin that returns the result of **SampleGeometricBag** on that uniform PSRN.
4. Set *ky* to $k * k / y$.
5. (At this point, we simulate $\exp(-U^2/y)$, $\exp(-k^2/y)$, $\exp(-U*k*2/y)$, as well as a scaled-down version of $U + k$, where U is the number built up by the uniform PSRN.) Call the **exp(−x/y) algorithm** with $x/y = ky$, then call the **exp(−(λ^k * x)) algorithm** using the input coin from step 2, $x = 1/y$, and $k = 2$, then call the same algorithm using the same input coin, $x = k * 2 / y$, and $k = 1$, then call the **sub-algorithm** given later with the uniform PSRN and $k = k$. If all of these calls return 1, the uniform PSRN was accepted. Otherwise, remove all digits from the uniform PSRN's fractional part and go to step 4.
6. If the uniform PSRN, call it *ret*, was accepted by step 5, fill it with uniform random digits as necessary to give its fractional part the desired number of digits (similarly to **FillGeometricBag**), and return $k + ret$.

The sub-algorithm below simulates a probability equal to $(U+k)/base^z$, where U is the number built by the uniform PSRN, *base* is the base (radix) of digits stored by that PSRN, *k* is an integer 0 or greater, and *z* is the number of significant digits in *k* (for this purpose, *z* is 0 if *k* is 0).

For base 2:

1. Set *N* to 0.
2. With probability 1/2, go to the next step. Otherwise, add 1 to *N* and repeat this step.
3. If *N* is less than *z*, return $\text{rem}(k / 2^{z - 1 - N}, 2)$. (Alternatively, shift *k* to the right, by $z - 1 - N$ bits, then return *k AND* 1, where "*AND*" is a bitwise AND-operation.)
4. Subtract *z* from *N*. Then, if the item at position *N* in the uniform PSRN's fractional

part (positions start at 0) is not set to a digit (e.g., 0 or 1 for base 2), set the item at that position to a digit chosen uniformly at random (e.g., either 0 or 1 for base 2), increasing the capacity of the uniform PSRN's fractional part as necessary.

5. Return the item at position *N*.

For bases other than 2, such as 10 for decimal, this can be implemented as follows (based on **URandLess**):

1. Set *i* to 0.
2. If *i* is less than *z*:
   1. Set *da* to rem(k / 2^{z-1-i}, *base*), and set *db* to a digit chosen uniformly at random (that is, an integer in the interval [0, *base*)).
   2. Return 1 if *da* is less than *db*, or 0 if *da* is greater than *db*.
3. If *i* is *z* or greater:
   1. If the digit at position ($i - z$) in the uniform PSRN's fractional part is not set, set the item at that position to a digit chosen uniformly at random (positions start at 0 where 0 is the most significant digit after the point, 1 is the next, etc.).
   2. Set *da* to the item at that position, and set *db* to a digit chosen uniformly at random (that is, an integer in the interval [0, *base*)).
   3. Return 1 if *da* is less than *db*, or 0 if *da* is greater than *db*.
4. Add 1 to *i* and go to step 3.

2.2 Uniform Distribution Inside a Circle

The following algorithm is an arbitrary-precision sampler for generating a point uniformly at random inside a circle centered at (0, 0) and with radius 1. It adapts the well-known rejection technique of generating X and Y coordinates until $X^2+Y^2 < 1$ (e.g., (Devroye 1986, p. 230 et seq.)**[(2)]**).

1. Generate two empty PSRNs, call them *x* and *y*, with a positive sign, an integer part of 0, and an empty fractional part.
2. Set *c* to *base*, where *base* is the base of digits to be stored by the PSRNs (such as 2 for binary or 10 for decimal). Then set *xd* to 0, *yd* to 0, and *d* to 1.
3. Multiply *xd* by *base* and add a digit chosen uniformly at random to *xd*. Then multiply *yd* by *base* and add a digit chosen uniformly at random to *yd*.
4. Set *lb* to *xd***xd* + *yd***yd*, and set *ub* to (*xd* + 1) * (*xd* + 1) + (*yd* + 1) * (*yd* + 1). (Here, *lb* and *ub* are lower and upper bounds, respectively, of the distance from the point (*xd*, *yd*) to the origin, scaled to c^2 units. These bounds can prove useful not just for implementing the uniform distribution inside a circle, but also the uniform distribution inside a shell or a set of concentric shells.)
5. If $ub < c^2$, then *xd* and *yd* lie inside the circle and are accepted. If they are accepted this way, then at this point, *xd* and *yd* will each store the *d* digits of a coordinate in the circle, expressed as a number in the interval [0, 1], or more precisely, a range of numbers. (For example, if *base* is 10, *d* is 3, and *xd* is 342, then the X-coordinate is 0.342, or more precisely, a number in the interval [0.342, 0.343].) In this case, do the following:
   1. Transfer the digits of *xd* and *yd* to *x*'s and *y*'s fractional parts, respectively. The variable *d* tells how many digits to transfer this way. (For example, if *base* is 10, *d* is 3, and *xd* is 342, set *x*'s fractional part to [3, 4, 2].)
   2. Fill *x* and *y* each with uniform random digits as necessary to give its fractional part the desired number of digits (similarly to **FillGeometricBag**).
   3. With probability 1/2, set *x*'s sign to negative. Then with probability 1/2, set *y*'s sign to negative.
   4. Return *x* and *y*, in that order.

6. If *lb* > c^2, then the point lies outside the circle and is rejected. In this case, go to step 2.
7. At this point, it is not known whether *xd* and *yd* lie inside the circle, so multiply *c* by *base*, then 1 to *d*, then go to step 3.

2.3 Sum of Exponential Random Numbers

An arbitrary-precision sampler for the sum of *n* exponential random numbers (also known as the Erlang(*n*) or gamma(*n*) distribution) is doable via partially-sampled uniform random numbers, though it is obviously inefficient for large values of *n*.

1. Generate *n* uniform PSRNs, and turn each of them into an exponential random number with a rate of 1, using an algorithm that employs rejection from the uniform distribution (such as the von Neumann algorithm or Karney's improvement to that algorithm (Karney 2014)**(3)**). This algorithm won't work for exponential PSRNs (e-rands), described in my article on **partially-sampled random numbers**, because the sum of two e-rands may follow a subtly wrong distribution. By contrast, generating exponential random numbers via rejection from the uniform distribution will allow unsampled digits to be sampled uniformly at random without deviating from the exponential distribution.
2. Generate the sum of the random numbers generated in step 1 by applying the **algorithm to add two PSRNs** given in another document.

2.4 Hyperbolic Secant Distribution

The following algorithm adapts the rejection algorithm from p. 472 in (Devroye 1986)**(2)** for arbitrary-precision sampling.

1. Generate an exponential PSRN, call it *ret*.
2. Set *ip* to 1 plus *ret*'s integer part.
3. (The rest of the algorithm accepts *ret* with probability 1/(1+*ret*).) With probability *ip*/(1+*ip*), generate a number that is 1 with probability 1/*ip* and 0 otherwise. If that number is 1, *ret* was accepted, in which case fill it with uniform random digits as necessary to give its fractional part the desired number of digits (similarly to **FillGeometricBag**), and return either *ret* or −*ret* with equal probability.
4. Call **SampleGeometricBag** on *ret*'s fractional part (ignore *ret*'s integer part and sign). If the call returns 1, go to step 1. Otherwise, go to step 3.

2.5 Mixtures

A *mixture* involves sampling one of several distributions, where each distribution has a separate probability of being sampled. In general, an arbitrary-precision sampler is possible if all of the following conditions are met:

- There is a finite number of distributions to choose from.
- The probability of sampling each distribution is a rational number, or it can be expressed as a function for which a **Bernoulli factory algorithm** exists.
- For each distribution, an arbitrary-precision sampler exists.

One example of a mixture is two beta distributions, with separate parameters. One beta distribution is chosen with probability exp(−3) (a probability for which a Bernoulli factory algorithm exists) and the other is chosen with the opposite probability. For the two beta distributions, an arbitrary-precision sampling algorithm exists (see my article on **partially-sampled random numbers** for details).

3 Notes

- [1] Fishman, D., Miller, S.J., "Closed Form Continued Fraction Expansions of Special Quadratic Irrationals", ISRN Combinatorics Vol. 2013, Article ID 414623 (2013).
- [2] Devroye, L., ***Non-Uniform Random Variate Generation***, 1986.
- [3] Karney, C.F.F., "**Sampling exactly from the normal distribution**", arXiv:1303.6257v2 [physics.comp-ph], 2014.

4 License