Randomized Estimation Algorithms

This version of the document is dated 2021-05-09.

Peter Occil

1 Introduction

This page presents general-purpose algorithms for estimating the mean value of a stream of random numbers, or estimating the mean value of a function of those numbers. The estimates are either *unbiased* (they have no systematic bias from the true mean value), or they come close to the true value with a user-specified error tolerance.

The algorithms are described to make them easy to implement by programmers.

2 Concepts

The following concepts are used in this document.

Each algorithm takes a stream of random numbers. These numbers follow a *probability distribution* or simply *distribution*, or a rule that says which kinds of numbers are more likely to occur than others. A distribution has the following properties.

- The *expectation*, *expected value*, or *mean* is the average value of the distribution. It is expressed as $\mathbf{E}[X]$, where X is a random number from the stream. In other words, take random samples and then take their average. The average will approach the expected value as n gets large.
- An n^{th} *moment* is the expected value of X^n. In other words, take random samples, raise them to the power n, then take their average. The average will approach the n^{th} moment as n gets large.
- An n^{th} *central moment (about the mean)* is the expected value of $(X^n - \mu)$, where μ is the distribution's mean. The 2nd central moment is called *variance*, and the 4th central moment *kurtosis*.
- An n^{th} *central absolute moment* (c.a.m.) is the expected value of $\text{abs}(X^n - \mu)$, where μ is the distribution's mean. This is the same as the central moment when n is even.

Some distributions don't have an n^{th} moment for a particular n. This usually means the n^{th} power of the random numbers varies so wildly that it can't be estimated accurately. If a distribution has an n^{th} moment, it also has a k^{th} moment for any k in the interval $1, n)$.

For any estimation algorithm, the *relative error* is $\text{abs}(est, trueval) - 1$, where *est* is the estimate and *trueval* is the true expected value.

3 A Relative-Error Algorithm for Bernoulli Random Numbers

The following algorithm from Huber (2017)[(1)] estimates the probability that a stream of random zeros and ones produces the number 1. The algorithm's relative error is

independent of that probability, however, and the algorithm produces *unbiased* estimates. Specifically, the stream of numbers has the following properties:

- The stream produces only zeros and ones (that is, the stream follows the **Bernoulli distribution**).
- The stream of numbers can't take on the value 0 with probability 1.
- The stream's mean (expected value) is unknown.

The algorithm, also known as *Gamma Bernoulli Approximation Scheme*, has the following parameters:

- ε, δ: Both parameters must be greater than 0, and ε must be 3/4 or less, and δ must be 1 or less.

With this algorithm, the relative error will be no greater than ε with probability $1 - \delta$ or greater. However, the estimate can be higher than 1 with probability greater than 0.

The algorithm, called **Algorithm A** in this document, follows.

1. Calculate the minimum number of samples k. There are two suggestions. The simpler one is k = ceil(-6*ln(2/δ)/(ε^2*(4*$\varepsilon-3$))). A more complicated one is the smallest integer k such that gammainc(k,($k-1$)/(1+ε)) + (1 − gammainc(k, ($k-1$)/(1−ε))) ≤ δ, where gammainc is the regularized lower incomplete gamma function.
2. Take samples from the stream until k 1's are taken this way. Let r be the total number of samples taken this way.
3. Generate g, a gamma(r) random variate, then return ($k-1$)/g.

   Note:

   1. As noted in Huber 2017, if we have a stream of random numbers that take on values in the interval [0, 1], but have unknown mean, we can transform each number by—

      1. generating a uniform(0, 1) random variate u, then
      2. changing that number to 1 if u is less than that number, or 0 otherwise,

      and we can use the new stream of zeros and ones in the algorithm to get an unbiased estimate of the unknown mean.

   2. As can be seen in Feng et al. (2016)[(2)], the following is equivalent to steps 2 and 3 of *Algorithm A*: "Let G be 0. Do this k times: 'Flip a coin until it shows heads, let r be the number of flips (including the last), and add a gamma(r) random variate to G.' The estimated probability of heads is then ($k-1$)/G.", and the following is likewise equivalent if the stream of random numbers follows a (zero-truncated) "geometric" distribution with unknown mean: "Let G be 0. Do this k times: 'Take a sample from the stream, call it r, and add a gamma(r) random variate to G.' The estimated mean is then ($k-1$)/G." (This is with the understanding that the geometric distribution is defined differently in different academic works.) The geometric algorithm produces unbiased estimates just like *Algorithm A*.

4 A Relative-Error Algorithm for Bounded Random Numbers

The following algorithm comes from Huber and Jones (2019)[3]; see also Huber (2017)[4]. It estimates the expected value of a stream of random numbers with the following properties:

- The numbers in the stream lie in the closed interval [0, 1].
- The stream of numbers can't take on the value 0 with probability 1.
- The stream's mean (expected value) is unknown.

The algorithm has the following parameters:

- ε, δ: Both parameters must be greater than 0, and ε must be 1/8 or less, and δ must be 1 or less. The relative error is abs(*est*, *trueval*) − 1, where *est* is the estimate and *trueval* is the true expected value.

With this algorithm, the relative error will be no greater than ε with probability $1 - \delta$ or greater. However, the estimate has a nonzero probability of being higher than 1.

The algorithm, called **Algorithm B** in this document, follows.

1. Set k to ceil(2*ln(6/δ)/$\varepsilon^{2/3}$).
2. Set b to 0 and n to 0.
3. (Stage 1: Modified gamma Bernoulli approximation scheme.) While b is less than k:
    1. Add 1 to n.
    2. Take a sample from the stream, call it s.
    3. Generate a uniform(0, 1) random number, call it u.
    4. If u is less than s, add 1 to b.
4. Set gb to k + 2, then divide gb by a gamma(n) random variate.
5. (Find the sample size for the next stage.) Set $c1$ to 2*ln(3/δ).
6. Set n to a Poisson($c1$/(ε*gb)) random variate.
7. Run the standard deviation sub-algorithm (given later) n times. Set A to the number of 1's returned by that sub-algorithm this way.
8. Set $csquared$ to (A / $c1$ + 1 / 2 + sqrt(A / $c1$ + 1 / 4)) * (1 + $\varepsilon^{1 / 3}$)^2*ε/gb.
9. Set n to ceil((2*ln(6/δ)/ε^2)/(1−$\varepsilon^{1/3}$)).
10. (Stage 2: Light-tailed sample average.) Set $e0$ to $\varepsilon^{1/3}$.
11. Set $mu0$ to gb/(1−$e0^2$).
12. Set $alpha$ to ε/($csquared$*$mu0$).
13. Set w to n*$mu0$.
14. Do the following n times:
    1. Get a sample from the stream, call it g. Set s to $alpha$*(g−$mu0$).
    2. If s≥0, add ln(1+s+s*s/2)/$alpha$ to w. Otherwise, subtract ln(1−s+s*s/2)/$alpha$ from w.
15. Return w/n.

The standard deviation sub-algorithm follows.

1. Generate an unbiased random bit. If that bit is 1 (which happens with probability 1/2), return 0.
2. Get two samples from the stream, call them x and y.
3. Generate a uniform(0, 1) random number, call it u.
4. If u is less than (x−y)^2, return 1. Otherwise, return 0.

Note: As noted in Huber and Jones, if the stream of random numbers takes on values in the interval [0, m], where m is a known number, we can divide the stream's numbers by m before using them in *Algorithm B*, and the algorithm will still work.

5 An Absolute-Error Adaptive Algorithm

The following algorithm comes from Kunsch et al. (2019)**(5)**. It estimates the mean of a stream of random numbers with the following properties:

- The distribution of numbers in the stream has a finite q^{th} c.a.m. and p^{th} c.a.m. (also called q-moment and p-moment, respectively, in this section).
- The exact q-moment and p-moment need not be known in advance.
- The q-moment's q^{th} root is no more than κ times the p-moment's p^{th} root, where κ is 1 or greater. (Note that the q-moment's q^{th} root is also known as *standard deviation* if $q = 2$, and *mean deviation* if $q = 1$; similarly for p.)

The algorithm works by first estimating the p-moment of the stream, then using the estimate to determine a sample size for the next step, which actually estimates the stream's mean.

The algorithm has the following parameters:

- ε, δ: Both parameters must be greater than 0, and δ must be 1 or less. The algorithm will return an estimate within ε of the true expected value with probability $1 - \delta$ or greater, and the estimate will not go beyond the bounds of the stream's numbers. The algorithm is not guaranteed to maintain a finite mean squared error or expected error in its estimates.
- p: The degree of the p-moment that the algorithm will estimate to determine the mean.
- q: The degree of the q-moment. q must be greater than p.
- κ: May not be less than the q-moment's q^{th} root divided by the p-moment's p^{th} root, and may not be less than 1.

For example:

- With parameters $p = 2$, $q = 4$, $\varepsilon = 1/10$, $\delta = 1/16$, $\kappa = 1.1$, the algorithm assumes the random numbers' distribution has a bounded 4th c.a.m. and that the 4th c.a.m.'s 4th root is no more than 1.1 times the 2nd c.a.m.'s square root (that is, the standard deviation), and will return an estimate that's within 1/10 of the true mean with probability $(1 - 1/16)$ or greater, or 15/16 or greater.
- With parameters $p = 1$, $q = 2$, $\varepsilon = 1/10$, $\delta = 1/16$, $\kappa = 2$, the algorithm assumes the random numbers' distribution has a standard deviation (q=2) that is no more than 2 times its mean deviation (p=1), and will return an estimate that's within 1/10 of the true mean with probability $(1 - 1/16)$ or greater, or 15/16 or greater.

The algorithm, called **Algorithm C** in this document, follows.

1. If κ is 1:
    1. Set n to ceil(ln(1/δ)/ln(2))+1.
    2. Get n samples from the stream and return (mn + mx)/2, where mx is the highest sample and mn is the lowest.
2. Set k to ceil((2*ln(1/δ))/ln(4/3)). If k is even, add 1 to k.
3. Set kp to k.
4. Set κ to $\kappa^{(p*q/(q-p))}$.
5. If q is 2 or less:
    - Set m to ceil($3*\kappa*48^{1/(q-1)}$); set s to $1+1/(q-1)$; set η to $16^{1/(q-1)}*\kappa/\varepsilon^s$.
6. If q is greater than 2:
    - Set m to ceil($144*\kappa$); set s to 2; set η to $16*\kappa/\varepsilon^s$.

7. (Stage 1: Estimate p-moment to determine number of samples for stage 2.) Create k many blocks. For each block:
    1. Get m samples from the stream.
    2. Add the samples and divide by m to get this block's sample mean, *mean*.
    3. Calculate the p-moment estimate for this block, which is: $(\sum_{i = 0, \ldots, k-1} (block[i] - mean)^p)/m$, where $block[i]$ is the sample at position i of the block (positions start at 0).
8. (Find the median of the p-moment estimates.) Sort the p-moment estimates from step 7 in ascending order, and set *median* to the value in the middle of the sorted list (at position floor(k/2) with positions starting at 0); this works because k is odd.
9. (Calculate sample size for the next stage.) Set mp to max(1, ceil($\eta * median^s$)).
10. (Stage 2: Estimate of the sample mean.) Create kp many blocks. For each block:
    1. Get mp samples from the stream.
    2. Add the samples and divide by mp to get this block's sample mean.
11. (Find the median of the sample means. This is definitely an unbiased estimate of the mean when kp is 1 or 2, but unfortunately, it isn't one for any $kp > 2$.) Sort the sample means from step 10 in ascending order, and return the value in the middle of the sorted list (at position floor(kp/2) with positions starting at 0); this works because kp is odd.

    Notes:

    1. If the stream of random numbers meets the condition for *Algorithm C* for a given q, p, and κ, then it still meets that condition when those numbers are multiplied by a constant or a constant is added to them.
    2. Theorem 3.4 of Kunsch et al. (2019)**(5)** shows that there is no mean estimation algorithm that—
        - produces an estimate within a user-specified error tolerance (in terms of *absolute error*, as opposed to *relative error*) with probability greater than a user-specified value, and
        - works for all streams whose distribution is known only to have a finite n^{th} moment for any fixed n (such as a finite mean or finite variance).

    Example: To estimate the probability of heads of a coin that produces either 1 with an unknown probability in the interval [μ, 1$-\mu$], or 0 otherwise, we can take $q = 4$, $p = 2$, and $\kappa \geq (1/\min(\mu, 1-\mu))^{1/4}$ (Kunsch et al. 2019, Lemma 3.6).

6 Estimating a Function of the Mean

Algorithm C can be used to estimate a function of the mean of a stream of random numbers with unknown mean. Specifically, the goal is to estimate $f(\mathbf{E}[\mathbf{z}])$, where:

- $\mathbf{z}$ is a random number produced by the stream. Each number produced by the stream must lie in the interval [0, 1].
- f is a known continuous function that maps the closed interval [0, 1] to [0, 1].
- The stream's numbers can take on a single value with probability 1.

The following algorithm takes the following parameters:

- p, q, and κ are as defined in *Algorithm C*.
- ε, δ: The algorithm will return an estimate within ε of $f(\mathbf{E}[\mathbf{z}])$ with probability $1 - \delta$ or greater, and the estimate will be in the interval [0, 1].

The algorithm, like *Algorithm C*, works only if the stream's distribution has the following

technical property: The q-th c.a.m.'s q-th root may not be less than κ times the p-th c.a.m.'s p-th root. The algorithm, called **Algorithm D** in this document, follows.

1. Calculate γ as a number equal to or less than $\psi(\varepsilon)$, which is found by taking the so-called *modulus of continuity* of $f(x)$, call it $\omega(\eta)$, and solving the equation $\omega(\eta) = \varepsilon$ for η.
    - Loosely speaking, a modulus of continuity shows the maximum range of f in a window of size η.
    - For example, if f's slope is continuous at every point and never vertical, then f is *Lipschitz continuous* and its modulus of continuity is $\omega(\eta) = M*\eta$, where M is the Lipschitz constant, which in this case is the maximum absolute value of f's "slope function". The solution for ψ is then $\psi(\varepsilon) = \varepsilon/M$.
    - Because f is continuous on a closed interval, it's guaranteed to have a modulus of continuity (by the Heine–Cantor theorem; see also a **related question**).
2. Run *Algorithm C* with the given parameters p, q, κ, and δ, but with $\varepsilon = \gamma$. Let μ be the result.
3. Return $f(\mu)$.

A simpler version of *Algorithm D* was given as an answer to the linked-to question. As with *Algorithm D*, this algorithm will return an estimate within ε of $f(\mathbf{E}[\mathbf{z}])$ with probability $1 - \delta$ or greater, and the estimate will be in the interval [0, 1]. The algorithm, called **Algorithm E** in this document, follows.

1. Calculate γ as given in step 1 of *Algorithm D*.
2. (Calculate the sample size.) Set n to ceil(ln(2/δ))/(2*γ^2). (As the answer notes, this sample size is based on Hoeffding's inequality.)
3. (Calculate the sample mean.) Get n samples from the stream, sum them, then divide the sum by n, then call the result μ. Return $f(\mu)$.

Notes:

1. *Algorithm D* and *Algorithm E* won't work in general when $f(x)$ has jump discontinuities (this happens in general when f is piecewise continuous, or made up of independent continuous pieces that cover all of [0, 1]), at least when ε is equal to or less than the maximum jump among all the jump discontinuities (see also a **related question**).

2. *Algorithm E* can be used to build so-called "**approximate Bernoulli factories**", or algorithms that approximately sample the probability $f(\lambda)$ given a coin with probability of heads of λ. In this case, the stream of numbers should produce only zeros and ones (and thus follow the *Bernoulli distribution*, and f should also be a continuous function. The approximate Bernoulli factory would work as follows: After running *Algorithm E* and getting an estimate, generate a uniform random number in [0, 1), then return 1 the number is less than the estimate, or 0 otherwise.

3. *Algorithm D* and *Algorithm E* can be adapted to apply to streams outputting numbers in a bounded interval [a, b] (where a and b are known rational numbers), but with unknown mean, as follows:

    - For each number in the stream, subtract a from it, then divide it by ($b - a$).
    - Instead of ε, take $\varepsilon/(b - a)$.
    - Instead of $f(x)$, take $g(x)$ where $g(x) = f(a + (x*(b - a)))$.
    - If *Algorithm D* or *Algorithm E* would return a result r, it returns $a + (r*(b - a))$ instead.

Examples:

1. Take $f(x) = \sin(\pi*x*4)/2 + 1/2$. This is a Lipschitz continuous function with Lipschitz constant $2*\pi$, so for this f, $\psi(\varepsilon) = \varepsilon/(2*\pi)$. Now, if we have a coin that produces heads with an unknown probability in the interval $[\mu, 1-\mu]$, or 0 otherwise, we can run *Algorithm D* or *Algorithm E* with $q = 4$, $p = 2$, and $\kappa \geq (1/\min(\mu, 1-\mu))^{1/4}$ (see the section on *Algorithm C*).
2. Take $f(x) = x$. This is a Lipschitz continuous function with Lipschitz constant 1, so for this f, $\psi(\varepsilon) = \varepsilon/1$.

7 Randomized Integration

Monte Carlo integration is a randomized way to estimate the integral of a function. *Algorithm C* can be used to estimate an integral of a function $h(\mathbf{z})$, with the following properties:

- $h(\mathbf{x})$ is a multidimensional function that takes an n-dimensional vector and returns a real number. $h(\mathbf{x})$ is usually a function that's easy to evaluate but whose integral is hard to calculate.
- $\mathbf{z}$ is an n-dimensional vector chosen at random in the sampling domain.

The estimate will come within ε of the true integral with probability $1 - \delta$ or greater, as long as the following conditions are met:

- The q^{th} c.a.m. for $h(\mathbf{z})$ is finite. That is, $\mathbf{E}[\text{abs}(h(\mathbf{z})-\mathbf{E}[h(\mathbf{z})])^q]$ is finite.
- The q^{th} c.a.m.'s q^{th} root is no more than κ times the p^{th} c.a.m.'s p^{th} root, where κ is 1 or greater.

Unfortunately, these conditions may be hard to verify in practice, especially when the distribution $h(\mathbf{z})$ is not known. (In fact, $\mathbf{E}[h(\mathbf{z})]$, as seen above, is the unknown integral that we seek to estimate.)

For this purpose, each number in the stream of random numbers is generated as follows (see also Kunsch et al.):

1. Set $\mathbf{z}$ to an n-dimensional vector (list of n numbers) chosen at random in the sampling domain, independently of any other choice. Usually, $\mathbf{z}$ is chosen *uniformly* at random this way.
2. Calculate $h(\mathbf{z})$, and set the next number in the stream to that value.

Alternatively, if $h(\mathbf{z})$ can take on only numbers in the closed interval [0, 1], the much simpler *Algorithm E* can be used on the newly generated stream (taking $f(x) = x$), rather than *Algorithm C*.

The following example (coded in Python for the SymPy computer algebra library) shows how to find parameter κ for estimating the integral of $\min(Z1, Z2)$ where *Z1* and *Z2* are each uniformly chosen at random in the interval [0, 1]. It assumes $p = 2$ and $q = 4$. (This is a trivial example because we can calculate the integral directly — 1/3 — but it shows how to proceed for more complicated cases.)

```
<h1>Distribution of Z1 and Z2</h1>

u1=Uniform('U1',0,1)
u2=Uniform('U2',0,1)
<h1>Function to estimate</h1>
```

```
func = Min(u1,u2)
emean=E(func)
p = S(2) # Degree of p-moment
q = S(4) # Degree of q-moment
<h1>Calculate value for kappa</h1>

kappa = E(Abs(func-emean)**q)**(1/q) / E(Abs(func-emean)**p)**(1/p)
pprint(Max(1,kappa))
```

8 Notes

- [1] Huber, M., 2017. A Bernoulli mean estimate with known relative error distribution. Random Structures & Algorithms, 50(2), pp.173-182. (preprint in arXiv:1309.5413v2 [math.ST], 2015).
- [2] Feng, J. et al. “Monte Carlo with User-Specified Relative Error.” (2016).
- [3] Huber, Mark, and Bo Jones. "Faster estimates of the mean of bounded random variables." Mathematics and Computers in Simulation 161 (2019): 93-101.
- [4] Huber, Mark, "**An optimal(ε, δ)-approximation scheme for the mean of random variables with bounded relative variance**", arXiv:1706.01478, 2017.
- [5] Kunsch, Robert J., Erich Novak, and Daniel Rudolf. "Solvable integration problems and optimal sample size selection." Journal of Complexity 53 (2019): 40-67. Also in **https://arxiv.org/pdf/1805.08637.pdf** .

9 License