Randomized Estimation Algorithms

Peter Occil

Randomized Estimation Algorithms

This version of the document is dated 2023-06-17.

Peter Occil

1 Introduction

Suppose there is an endless stream of numbers, each generated at random and independently from each other, and as many numbers can be sampled from the stream as desired. These numbers are called *random variates*. This page presents general-purpose algorithms for estimating the mean value ("long-run average") of those variates, 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.

Not yet covered are the following algorithms:

- Unbiased mean estimation algorithms that take a sequence of estimators that get better and better at estimating the desired mean (for example, estimators that average an increasing number of sample points). See, for example, Vihola (2018)[1].

1.1 About This Document

This is an open-source document; for an updated version, see the source code[2] or its rendering on GitHub[3]. You can send comments on this document on the GitHub issues page[4].

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[5]**:

- 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.

[1] Vihola, M., 2018. Unbiased estimators and multilevel Monte Carlo. Operations Research, 66(2), pp.448-462.
[2] https://github.com/peteroupc/peteroupc.github.io/raw/master/estimation.md
[3] https://github.com/peteroupc/peteroupc.github.io/blob/master/estimation.md
[4] https://github.com/peteroupc/peteroupc.github.io/issues
[5] https://github.com/peteroupc/peteroupc.github.io/issues/18

2 Concepts

The following concepts are used in this document.

The *closed unit interval* (written as [0, 1]) means the set consisting of 0, 1, and every real number in between.

Each algorithm takes a stream of independent random variates (numbers). These variates follow a *probability distribution* or simply *distribution*, or a rule that says which kinds of numbers have greater probability of occurring than others. A distribution has the following properties.

- The *expectation*, *expected value*, or *mean* is the "long-run average" value of the distribution. It is expressed as $\mathbf{E}[X]$, where X is a number taken from the stream. If $\mathbf{E}[X]$ exists, then with probability 1, the average of the first n sampled items taken from the stream approaches the expected value as n gets large (as a result of the *law of large numbers*).
- An n^{th} *moment* is the expected value of X^n.
- An n^{th} *central moment (about the mean)* is the expected value of $(X - \mu)^n$, where μ is the distribution's mean. The 2nd central moment is called *variance*.
- An n^{th} *central absolute moment* (c.a.m.) is the expected value of $\text{abs}(X - \mu)^n$, 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 stream's 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 every integer k greater than 0 and less than n.

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

3 A Relative-Error Algorithm for a Bernoulli Stream

The following algorithm from Huber (2017)[6] 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 less than 1.

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 $k = \text{ceil}(-6*\ln(2/\delta)/(\varepsilon^2*(4*\varepsilon - 3)))$. A more complicated one is the smallest integer k such that $\text{gammainc}(k,(k-1)/(1+\varepsilon)) + (1 - \text{gammainc}(k,(k-1)/(1-\varepsilon))) \leq \delta$, 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 random variate with shape parameter r and scale 1, then return $(k-1)/g$.

Notes:

[6] 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).

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

    1. generating u, a uniform random variate between 0 and 1, 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)[7], 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), generate a gamma random variate with shape parameter r and scale 1, and add that variate to G.' The estimated probability of heads is then $(k-1)/\mathrm{G}$.", and the following is likewise equivalent if the stream of random variates 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, generate a gamma random variate with shape parameter r and scale 1, and add that variate to G.' The estimated mean is then $(k-1)/\mathrm{G}$." (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*.

3. The generation of a gamma random variate and the division by that variate can cause numerical errors in practice, such as rounding and cancellations, unless care is taken.

4. Huber proposes another algorithm that claims to be faster when the mean is bounded away from zero; see (Huber 2022)[8].

4 A Relative-Error Algorithm for a Bounded Stream

The following algorithm comes from Huber and Jones (2019)[9]; see also Huber (2017)[10]. It estimates the expected value of a stream of random variates with the following properties:

- The numbers in the stream lie in the closed unit interval.
- 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 less than 1.

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.

This algorithm is not guaranteed to produce unbiased estimates.

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

1. Set k to ceil(2*ln(6/ δ)/ ε $^{2/3}$).
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.

[7]Feng, J. et al. "Monte Carlo with User-Specified Relative Error." (2016).

[8]Huber, M., "**Tight relative estimation in the mean of Bernoulli random variables**", arXiv:2210.12861 [cs.LG], 2022. https://arxiv.org/abs/2210.12861

[9]Huber, Mark, and Bo Jones. "Faster estimates of the mean of bounded random variables." Mathematics and Computers in Simulation 161 (2019): 93-101.

[10]Huber, Mark, "**An optimal(ε , δ)-approximation scheme for the mean of random variables with bounded relative variance**", arXiv:1706.01478, 2017. https://arxiv.org/abs/1706.01478

3. With probability s (for example, if a newly generated uniform random variate between 0 and 1 is less than s), add 1 to b.
4. Set gb to $k + 2$, then generate a gamma random variate with shape parameter n and scale 1, then divide gb by that variate.
5. (Find the sample size for the next stage.) Set $c1$ to 2*ln(3/ δ).
6. Generate a Poisson random variate with mean $c1/(\ \varepsilon\ ^{*}gb)$, call it n.
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 $(A\ /\ c1 + 1\ /\ 2 + \mathrm{sqrt}(A\ /\ c1 + 1\ /\ 4))\ *\ (1 + \varepsilon^{1/3})^{2} *\ \varepsilon\ /gb$.
9. Set n to ceil((2*ln(6/ δ)/ ε^{2})/(1 − $\varepsilon^{1/3}$)), or an integer greater than this.
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 $\varepsilon\ /(csquared{*}mu0)$.
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 $s \geq 0$, 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 u, a uniform random variate between 0 and 1.
4. If u is less than $(x - y)^{2}$, return 1. Otherwise, return 0.

Notes:

1. As noted in Huber and Jones, if the stream of random variates 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.
2. While this algorithm is exact in theory (assuming computers can store real numbers of any precision), practical implementations of it can cause numerical errors, such as rounding and cancellations, unless care is taken.

5 An Absolute-Error Adaptive Algorithm

The following algorithm comes from Kunsch et al. (2019)[^7]. It estimates the mean of a stream of random variates 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.
- The exact q^{th} c.a.m. and p^{th} c.a.m. need not be known in advance.
- The q^{th} c.a.m.'s q^{th} root divided by the p^{th} c.a.m.'s p^{th} root is no more than κ, where κ is 1 or greater. (The q^{th} c.a.m.'s q^{th} root is also known as *standard deviation* if $q = 2$, and *mean absolute deviation* if $q = 1$; similarly for p.)

The algorithm works by first estimating the p^{th} c.a.m. of the stream, then using the estimate to determine a sample size for the next step, which actually estimates the stream's mean.

This algorithm is not guaranteed to produce unbiased estimates.

The algorithm has the following parameters:

- ε , δ : Both parameters must be greater than 0, and δ must be less than 1. 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^{th} c.a.m. that the algorithm will estimate to determine the mean.
- q: The degree of the q^{th} c.a.m. q must be greater than p.
- κ: Maximum value allowed for the following value: the q^{th} c.a.m.'s q^{th} root divided by the p^{th} c.a.m.'s p^{th} root. (If $p = 2$ and $q = 4$, this is the maximum value allowed for the kurtosis's 4th root (Hickernell et al. 2012)[11] [12].) κ may not be less than 1.

Both p and q must be 1 or greater and are usually integers.

For example:

- With parameters $p = 2$, $q = 4$, $\varepsilon = 1/10$, $\delta = 1/16$, $\kappa = 1.1$, the algorithm assumes the stream's numbers are distributed so that the kurtosis's 4th root, that is, the 4th c.a.m.'s 4th root (q=4) divided by the standard deviation (p=2), is no more than 1.1 (or alternatively, the kurtosis is no more than $1.1^4 = 1.4641$), and will return an estimate that's within 1/10 of the true mean with probability at least $(1 - 1/16)$ or 15/16.
- With parameters $p = 1$, $q = 2$, $\varepsilon = 1/10$, $\delta = 1/16$, $\kappa = 2$, the algorithm assumes the stream's numbers are distributed so that the standard deviation (q=2) divided by the mean deviation (p=1) is no more than 2, and will return an estimate that's within 1/10 of the true mean with probability at least $(1 - 1/16)$ or 15/16.

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 (or an integer greater than this).
    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[13], add 1 to k.
3. Set kp to k.
4. Set κ to $\kappa^{(/(-))}$.
5. If q is 2 or less:
    - Set m to ceil($3*\kappa*48^{1/(-1)}$) (or an integer greater than this); set s to $1+1/(q-1)$; set h to $16^{1/(-1)} *\kappa/\varepsilon^{s}$.
6. If q is greater than 2:
    - Set m to ceil($144*\kappa$); set s to 2; set h to $16*\kappa/\varepsilon^{s}$.
7. (Stage 1: Estimate p^{th} c.a.m. 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 estimate of the p^{th} c.a.m. for this block, which is: $((block[0] - mean)^p + block[1] - mean)^p + \ldots + block[k-1] - mean)^p)/m$, where $block[i]$ is the sample at position i of the block (positions start at 0).
8. (Find the median of the p^{th} c.a.m. estimates.) Sort the estimates calculated by 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($h * median^s$)), or an integer greater than this.
10. (Stage 2: Estimate of the sample mean.) Create kp many blocks. For each block:
    1. Get mp samples from the stream.

[11] Hickernell, F.J., Jiang, L., et al., "**Guaranteed Conservative Fixed Width Intervals via Monte Carlo Sampling**", arXiv:1208.4318v3 [math.ST], 2012/2013. https://arxiv.org/abs/1208.4318v3

[12] As used here, kurtosis is the 4th c.a.m. divided by the square of the 2nd c.a.m.

[13] "k is even" means that k is divisible by 2. This is true if $k - 2*\text{floor}(k/2)$ equals 0, or if the least significant bit of abs(x) is 0.

2. Add the samples and divide by *mp* to get this block's sample mean.

11. (Find the median of the sample means. It can be shown that this is an unbiased estimate of the mean when *kp* is 1 or 2, but not 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. The interval $[\hat{\mu}-\epsilon, \hat{\mu}+\epsilon]$ is also known as a *confidence interval* for the mean, with *confidence level* at least $1-\delta$ (where $\hat{\mu}$ is an estimate of the mean returned by *Algorithm C*).
2. If the stream of random variates meets the condition for *Algorithm C* for a given q, p, and κ, then it still meets that condition when those variates are multiplied by a constant or a constant is added to them.
3. Theorem 3.4 of Kunsch et al. (2019)[^7] shows that there is no mean estimation algorithm that—
   - produces an estimate within a user-specified error tolerance with probability greater than a user-specified value, and
   - works for all streams whose distribution is known only to have finite moments (the moments are bounded but the bounds are unknown).
4. There is also a mean estimation algorithm for very high dimensions, which works if the stream of multidimensional variates has a finite variance (Lee and Valiant 2022)[14], but this algorithm is impractical — it requires millions of samples at best.

Examples:

1. To estimate the probability of heads of a coin that produces either 1 with an unknown probability in the interval $[\ \mu\ ,\ 1-\mu\ ]$, 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).
2. The kurtosis of a Poisson distribution with mean μ is $(3 + 1/\ \mu\ )$. Thus, for example, to estimate the mean of a stream of Poisson variates with mean ν or greater but otherwise unknown, we can take $q = 4$, $p = 2$, and $\kappa \geq (3 + 1/\ \nu\ )^{\ 1/4}$.
3. The kurtosis of an exponential distribution is 9 regardless of its rate. Thus, to estimate the mean of a stream of exponential variates with unknown mean, we can take $q = 4$, $p = 2$, and $\kappa \geq 9^{\ 1/4} = \text{sqrt}(3)$.

6 Estimating the Mode

Suppose there is an endless stream of items, each generated at random and independently from each other, and we can sample as many items from the stream as we want. Then the following algorithm estimates the most frequently occurring item, called the *mode*.(Dutta and Goswami 2010)[15] This assumes the following are known:

- Exactly one item must occur more frequently than the others.
- ϵ is greater than 0 and less than one half of the smallest possible difference between the mode's probability and the next most frequent item's probability.
- δ is greater than 0 and less than 1.
- n is the number of distinct items that can be taken.

The following algorithm correctly estimates the mode with probability $1-\delta$.

[14] Lee, J.C. and Valiant, P., 2022. **Optimal Sub-Gaussian Mean Estimation in Very High Dimensions**. In 13th Innovations in Theoretical Computer Science Conference (ITCS 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik. https://drops.dagstuhl.de/opus/volltexte/2022/15694/

[15] Dutta, Santanu, and Alok Goswami. "Mode estimation for discrete distributions." Mathematical Methods of Statistics 19, no. 4 (2010): 374-384.

1. Calculate $m = \text{ceil}(\frac{(4\epsilon+3)(\ln(\frac{n}{\delta})+\ln(2))}{6\epsilon^2})$.
2. Take m items from the stream. If one item occurs more frequently than any other item taken this way, return the most frequent item. Otherwise, return an arbitrary but fixed item (among the items the stream can take).

7 Estimating a Function of the Mean

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

- $\mathbf{z}$ is a number produced by the stream.
- f is a known continuous function.
- 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 closed unit interval.

The algorithm works only if:

- Each number produced by the stream $\mathbf{z}$ satisfies $0 \leq \mathbf{z} \leq 1$.
- $f(x)$ maps the closed unit interval to itself.
- Like *Algorithm C*, the q^{th} c.a.m.'s q^{th} root divided by the p^{th} c.a.m.'s p^{th} root is no more than κ, where κ is 1 or greater.

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

1. Calculate γ as a number equal to or less than (ε), or the *inverse modulus of continuity*, which is found by taking the so-called *modulus of continuity* of $f(x)$, call it $\omega (h)$, and solving the equation $\omega (h) = \varepsilon$ for h.
   - Loosely speaking, a modulus of continuity $\omega (h)$ gives the maximum range of f in a window of size h.
   - For example, f is *Lipschitz continuous* with Lipschitz constant M or less[16], its modulus of continuity is $\omega (h) = M*h$. The solution for is then $(\varepsilon) = \varepsilon / M$.
   - 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**[17]).
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* takes the sample mean as the basis for the randomized estimate, that is, n samples are taken from the stream and averaged. As with *Algorithm D*, the following algorithm will return an estimate within ε of $f(\mathbf{E}[\mathbf{z}])$ with probability $1 - \delta$ or greater, and the estimate will not go beyond the bounds of the stream's numbers. 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.) Calculate the sample size n, depending on the distribution the stream takes and the function f.
3. (Calculate f of the sample mean.) Get n samples from the stream, sum them, then divide the sum by n, then call the result μ . Return $f(\mu)$.

[16] A *Lipschitz continuous* function with Lipschitz constant M is a continuous function f such that $f(x)$ and $f(y)$ are no more than $M* \delta$ apart whenever x and y are in the function's domain and no more than δ apart.Roughly speaking, the function's "steepness" is no greater than that of $M*x$.

[17] https://stats.stackexchange.com/questions/522429

Then the table below shows how the necessary sample size n can be determined.

Stream's distribution	Property of f	Sample size
Bounded; lies in the closed unit interval.[18]	Continuous; maps the closed unit interval to itself.	$n = \text{ceil}(\ln(2/\ \delta\ )/(2^*\ \gamma\ ^2))$.
Unbounded (can take on any real number) and has a known upper bound on the standard deviation σ (or the variance $\sigma\ ^2$).[19]	Bounded and continuous on every closed interval of the real line.	$n = \text{ceil}(\ \sigma\ ^2/(\ \delta\ ^*\ \gamma\ ^2))$.
Unbounded and subgaussian[20]; known upper bound on standard deviation σ (Wainwright 2019)[21]	$f(x) = x$.	$n = \frac{2\sigma^2 \ln(\frac{2}{\delta})}{\epsilon^2}$.

Notes:

1. *Algorithm D* and *Algorithm E* won't work in general when $f(x)$ has jump discontinuities (this can happen when f is only piecewise continuous, or made up of independent continuous pieces that cover f's whole domain), at least when ε is equal to or less than the maximum jump among all the jump discontinuities (see also a **related question**[22]).
2. If the input stream outputs numbers in a closed interval $[a,\ b]$ (where a and b are known rational numbers), but with unknown mean, and if f is a continuous function that maps the interval $[a,\ b]$ to itself, then *Algorithm D* and *Algorithm E* can be used as follows:
   - For each number in the stream, subtract a from it, then divide it by $(b - a)$.
   - Instead of ε , take $\varepsilon\ /(b - a)$.
   - Run either *Algorithm E* (calculating the sample size for the case "Bounded; lies in the closed unit interval") or *Algorithm D*. If the algorithm would return $f(\ \mu\ )$, instead return $g(\ \mu\ )$ where $g(\ \mu\ ) = f(a + (\ \mu\ ^*(b - a)))$.
3. *Algorithm E* is not an unbiased estimator in general. However, when $f(x) = x$, the sample mean used by the algorithm is an unbiased estimator of the mean as long as the sample size n is unchanged.
4. In *Algorithm E*, for an estimate in terms of relative error, rather than absolute error, multiply γ by M after step 1 is complete, where M is the smallest absolute value of the mean that the stream's distribution is allowed to have (Wainwright 2019)[23].
5. Finding the sample size needed to produce an estimate that is close to the true value with a user-specified error bound and a user-specified probability, as with *Algorithm E*, is also known as *offline learning*, *statistical learning*, or *provably approximately correct (PAC) learning*.

Examples:

[18] This was given as an **answer to a Stack Exchange question**; see also Jiang and Hickernell, "**Guaranteed Monte Carlo Methods for Bernoulli Random Variables**", 2014. As the answer notes, this sample size is based on Hoeffding's inequality. https://stats.stackexchange.com/questions/522429 https://arxiv.org/abs/1411.1151

[19] Follows from Chebyshev's inequality. The case of $f(x)$=x was mentioned as Equation 14 in Hickernell et al. (2012/2013).

[20] Roughly speaking, a distribution is *subgaussian* if the probability of taking on high values decays at least as fast as the normal distribution. In addition, every distribution taking on only values in a closed interval $[a,\ b]$ is subgaussian. See section 2.5 of R. Vershynin, *High-Dimensional Probability*, 2020.

[21] Wainwright, M.J., High-dimensional statistics: A non-asymptotic viewpoint, 2019.

[22] https://stats.stackexchange.com/questions/522429

[23] Wainwright, M.J., High-dimensional statistics: A non-asymptotic viewpoint, 2019.

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, $(\ \varepsilon\ ) = \varepsilon\ /(2*\ \pi\ )$. 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 the bounded case of *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, $(\ \varepsilon\ ) = \varepsilon\ /1$.
3. The variance of a Poisson distribution with mean μ is μ . Thus, for example, to estimate the mean of a stream of Poisson variates with mean ν or less but otherwise unknown, we can take $\sigma = \text{sqrt}(\ \nu\ )$ so that the sample size n is $\text{ceil}(\ \sigma\ ^2/(\ \delta\ *\ \varepsilon\ ^2))$, in accordance with the second case of *Algorithm E*.

8 Randomized Integration

Monte Carlo integration is a randomized way to estimate the integral ("area under the graph") of a function.[24]

This time, suppose there is an endless stream of *vectors* (d-dimensional points), each generated at random and independently from each other, and as many vectors can be sampled from the stream as wanted.

Suppose the goal is to estimate an integral of a function $h(\mathbf{z})$, where $\mathbf{z}$ is a vector from the stream, with the following properties:

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

Then *Algorithm C* will take the new stream and generate an estimate that comes 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 divided by the $p^{\ th}$ c.a.m.'s $p^{\ th}$ root is no more than κ, where κ is 1 or greater.

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

To use *Algorithm C* for this purpose, each number in the stream of random variates 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 (see note later in this section).
2. Calculate $h(\mathbf{z})$, and set the next number in the stream to that value.

   Example: 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 closed unit interval. It assumes $p = 2$ and $q = 4$. (This

[24] Deterministic (non-random) algorithms for integration or for finding the minimum or maximum value of a function are outside the scope of this article. But there are recent exciting developments in this field — see the following works and works that cite them:Y. Zhang, "Guaranteed, adaptive, automatic algorithms for univariate integration: methods, costs and implementations", dissertation, Illinois Institute of Technology, 2018.N. Clancy, Y. Ding, et al., The cost of deterministic, adaptive, automatic algorithms: cones, not balls. Journal of Complexity, 30(1):21–45, 2014.Mishchenko, Konstantin. "**Regularized Newton Method with Global $ O (1/k\hat{}2) $ Convergence**", arXiv:2112.02089 (2021).Doikov, Nikita, K. Mishchenko, and Y. Nesterov. "**Super-universal regularized Newton method**", arXiv:2208.05888 (2022). https://arxiv.org/abs/2112.02089 https://arxiv.org/abs/2208.05888

is a trivial example because the integral can be calculated directly — 1/3 — but it shows how to proceed for more complicated cases.)

```
# Distribution of Z1 and Z2
u1=Uniform('U1',0,1)
u2=Uniform('U2',0,1)
# Function to estimate
func = Min(u1,u2)
emean=E(func)
p = S(2) # Degree of p-moment
q = S(4) # Degree of q-moment
# Calculate value for kappa
kappa = E(Abs(func-emean)**q)**(1/q) / E(Abs(func-emean)**p)**(1/p)
pprint(Max(1,kappa))
```

Rather than *Algorithm C*, *Algorithm E* can be used (taking $f(x) = x$) if the distribution of $h(\mathbf{z})$, the newly generated stream, satisfies the properties given in the table for *Algorithm E*.

Note: If $h(\mathbf{z})$ is one-dimensional, maps the closed unit interval to itself, and is Lipschitz continuous with Lipschitz constant 1 or less, then the sample size for *Algorithm E* can be $n =$ ceil(23.42938/ ε ^{2}). (n is an upper bound calculated using Theorem 15.1 of Tropp (2021)[25], one example of a *uniform law of large numbers*). This sample size ensures an estimate of the integral with an expected absolute error of ε or less.

Note: As an alternative to the usual process of choosing a point uniformly in the *whole* sampling domain, *stratified sampling* (Kunsch and Rudolf 2018)[26], which divides the sampling domain in equally sized boxes and finds the mean of random points in those boxes, can be described as follows (assuming the sampling domain is the d-dimensional hypercube $[0, 1]^d$):

1. For a sample size n, set m to floor(n $^{1/}$), where d is the number of dimensions in the sampling domain (number of components of each point). Set s to 0.
2. For each *i[1]* in [0, m), do: For each *i[2]* in [0, m), do: ..., For each *i[d]* in [0, m), do:
    1. For each dimension j in [1, d], set *p[j]* to a number in the half-open interval [*i[j]*/m, (*i[j]*+1)/m) chosen uniformly at random.
    2. Add f((*p[1]*, *p[2]*, ..., *p[j]*)) to s.
3. Return s/m^d.

The paper (Theorem 3.9) also implied a sample size n for use in stratified sampling when f is Hölder continuous with Hölder exponent β or less [27] and is defined on the d-dimensional hypercube $[0, 1]^d$, namely $n =$ ceil((ln(2/ δ)/2* ε ^{2}) $^{/(2\beta+)}$).

9 Finding Coins with Maximum Success Probabilities

Given m coins each with unknown probability of heads, the following algorithm finds the k coins with the greatest probability of showing heads, such that the algorithm correctly finds them with probability at least $1 - \delta$. It uses the following parameters:

- k is the number of coins to return.

[25] J.A. Tropp, "ACM 217: Probability in High Dimensions", Caltech CMS Lecture Notes 2021-01, Pasadena, March 2021. Corrected March 2023.

[26] Kunsch, R.J., Rudolf, D., "**Optimal confidence for Monte Carlo integration of smooth functions**", arXiv:1809.09890, 2018. https://arxiv.org/abs/1809.09890

[27] 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^* δ $^{\alpha}$ 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^{α}. https://en.wikipedia.org/wiki/Hölder_condition

- δ is the confidence level; the algorithm correctly finds the coins with the greatest probability of showing heads with probability at least $1 - \delta$.
- D is a *gap parameter* or a lesser number, but must be greater than 0. The *gap parameter* is the difference between the k^{th} most probable coin to show heads and the $(k+1)^{th}$ most probable coin to show heads. Practically speaking, D is the smallest possible difference between one probability of heads and another.
- r is the number of rounds to run the algorithm and must be an integer 1 or greater.

In this section, ilog(a, r) means either a if r is 0, or max(ln(ilog(a, $r - 1$)), 1) otherwise.

Agarwal et al. (2017)[28] called this algorithm "aggressive elimination", and it can be described as follows.

1. Let t be ceil((ilog(m, r) + ln(8*k/ δ)) * 2/(D*D)).
2. For each integer i in [1, m], flip the coin labeled i, t many times, then set $P[i]$ to a list of two items: first is the number of times coin i showed heads, and second is the label i.
3. Sort the $P[i]$ in decreasing order by their values.
4. If r is 1, return the labels to the first k items in the list P, and the algorithm is done.
5. Set μ to ceil(k + m/ilog(m, $r - 1$)).
6. Let C be the coins whose labels are given in the first μ items in the list (these are the μ many coins found to be the most biased by this algorithm).
7. If $\mu \leq 2$*k, do a recursive run of this algorithm, using only the coins in C and with $\delta = \delta$ /2 and $r =$ 1.
8. If $\mu > 2$*k, do a recursive run of this algorithm, using only the coins in C and with $\delta = \delta$ /2 and $r = r - 1$.

10 Requests and Open Questions

For open questions, see "**Questions on Estimation Algorithms**[29]".

11 Notes

12 License

[28] Agarwal, A., Agarwal, S., et al., "Learning with Limited Rounds of Adaptivity: Coin Tossing, Multi-Armed Bandits, and Ranking from Pairwise Comparisons", *Proceedings of Machine Learning Research* 65 (2017).

[29] https://peteroupc.github.io/requestsother.html#Questions_on_Estimation_Algorithms

[30] https://creativecommons.org/publicdomain/zero/1.0/