Randomized Estimation Algorithms

Peter Occil

Randomized Estimation Algorithms

This version of the document is dated 2023-06-13.

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 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**.

[1]Vihola, M., 2018. Unbiased estimators and multilevel Monte Carlo. Operations Research, 66(2), pp.448-462.

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.

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 *nth moment* is the expected value of Xn.
- An *nth 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 *nth central absolute moment* (c.a.m.) is the expected value of 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 nth moment for a particular n. This usually means the nth power of the stream's numbers varies so wildly that it can't be estimated accurately. If a distribution has an nth moment, it also has a kth moment for every k in the interval [1, n).]_ 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/md. > > 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 [2] and is defined on

[2]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

the d-dimensional hypercube [0, 1]d, namely n = ceil((ln(2/ δ)/2* ε 2)d/(2* β +d)).

3 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.
- 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 kth 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)[3] 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 $\mu \le 2$*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 $r = r - 1$.

whenever x and y are in the function's domain and no more than δ apart.Here, α satisfies $0 < \alpha \le 1$.Roughly speaking, the function's "steepness" is no greater than that of M*x α .

[3]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).

4 Requests and Open Questions

For open questions, see “**Questions on Estimation Algorithms**”.

5 Notes

6 License