Practical Algorithms for Self Scaling Histograms
or
Better than Average Data Collection

Michael Greenwald
Computer Science Department
Stanford University

October 24th, 1995

Abstract

This paper presents practical algorithms for implementing self-scaling histograms. We show that these algorithms can deal well with observations drawn from either continuous or discrete distributions, have fixed storage and low computational overhead, and can faithfully capture the distribution of the data with very low error.

As a tool for intrusive large-scale performance measurement, histograms are the ideal compromise between practical limitations (storage, computational cost, interference with the system being measured) and the desire for a complete record of all observations. In practice they are infrequently used because histograms are perceived as cumbersome to use, and it is often hard to decide in advance on appropriate parameters (bucket sizes, range). Use of programming language technology (object-oriented techniques for example) can solve the first problem, and the algorithms presented here for self-scaling histograms can solve the second.

The intended goal of these algorithms is to facilitate the change of the most common method of metering from simple (but possibly misleading) averages to more informative histograms.

1 Introduction

Performance measurement of computer systems requires some degree of sophistication and some degree of care. Ideally, the measurement is performed as a scientific experiment, with respect to reproducibility, controls, isolation from interference, and comprehensive collection of data. In practice, however, most performance metering is done, by necessity, using crude tools, careless methodology, and "large scale" techniques.

The most common forms of metering are, arguably, (a) counters and meters inserted into long-running programs and operating system kernels, (b) metering using readily available, general purpose, performance evaluation tools (e.g. `prof` or `gprof` [8]), and (c) an initial round (often the only round!) of metering during the final phase of development of a computer program or system. Further, these approaches are commonly used to detect, diagnose, and correct specific performance problems — rather than to acquire data for modeling or analysis. These are examples of "large scale", or "coarse grained", metering.

Even in such situations the *ideal* act of metering would recover, with little or no perturbation of the system being metered, a complete timestamped record of *every* event of possible interest, with all analysis done on the complete trace after the fact. In the common cases described above this is impractical and the events must be summarized or compressed on-the-fly using some form of data collection. The most common form of data collection in these cases, despite the loss of information, the chance of being misled, and other known drawbacks, compresses all observations into one or two numbers, often the sample mean[1], intended to summarize all observations.

Recovering more information about the distribution of the data would be preferable, but due to memory and computational overhead constraints, and ease of use considerations, more accurate or informative techniques are relatively unused.

[1] Other examples are the median, or range (min, max) and/or standard deviation.

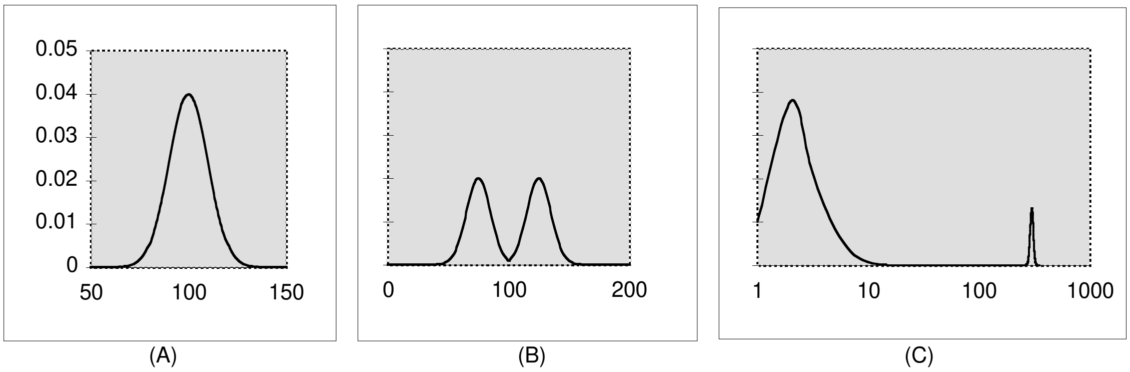


Figure 1: Different distributions (pdf), all with average value 100.0.

As a tool for intrusive *large-scale* performance measurement, histograms are the ideal compromise between practical limitations (storage, computational cost, interference with the system being measured) and the desire for a complete record of all observations.

In practice histograms are infrequently used because existing algorithms either require you to specify the histogram parameters in advance (often impossible, and always cumbersome), or have significant overhead, or reproduce the data poorly in cases of interest.

We present algorithms for two classes of self-scaling histograms that have lower computational and space overhead, greater accuracy, and are as easy to use as (or easier than) data structures implemented using existing algorithms. Further, we improve the performance and accuracy of a third class of histograms: equi-probable histograms using the P^2 algorithm[9].

These algorithms enable the typical programmer dashing off monitoring code to meter his or her program to replace the cliched use of `average`, `standard-deviation` and/or `count` with a more informative histogram that attempts to reproduce the entire distribution of values, rather than a single statistic.

Section 2 discusses the advantages of using histograms and the barriers which, to date, have restricted their wider deployment, section 3 describes algorithms for self-scaling histograms which most accurately and efficiently reconstruct the complete distribution of observations, section 4 addresses the common case of discrete distributions, section 5 presents efficient algorithms for histograms with equi-probable buckets, and section 6 presents the results of measuring the effectiveness of our algorithms. We then touch upon related work, and present our conclusions.

2 The Virtues of Histograms

Many statistics texts and texts on computer performance evaluation point out how inadequately a single statistic captures the full distribution.

Consider the three different distributions shown in Figure 1. All have the same average value, but their distributions are *very* different and represent quite different behaviors. Clearly the average value is not sufficient to distinguish between these cases, and will obscure our understanding of the program's actual behavior. This misperception can sometimes lead to an inappropriate plan to improve the performance of the system, or, at best, cause you to explore fruitless avenues of optimization.

Histograms are a way of recovering an approximation of the entire distribution. They are relatively efficient, require fixed storage, and provide substantially more information than, for example, the sample mean and standard deviation.

They are especially useful when you do not have a precise idea of what you are looking for, which is often the case in coarse-grained metering. Consider the following (trivial) examples where the results were precisely things *not* being searched for.

```
Avg: 2.65822  Std Dev: 3.03867182 (Avg clock overhead: .18786)
Low: 2,  High: 267, Count: 50,000

  Main Mode:
  Avg: 2.6207593    Stdev: 0.48523931
  Low: 2   High: 4  Bucket-size: 1  Count:   49,992

  2: 18960   3: 31031    4: 1

     Overflow:
     Low: 171  High: 267  Count: 8
     Avg: 236.75   Std Dev:  37.821125

     Bucket       Count    Bucket        Count    Bucket        Count
    171-190:      2        211-230:               251-270:      6
    191-210:               231-250:

     Exact data available for overflow: (257 258 256 257 172 267 171 256)
```

Table 1: Histogram of results of measuring a single array-reference with interrupts masked.

2.1 Examples

2.1.1 Sequence Break

Table 1 contains the results of timing[2] many iterations of a single array reference with interrupts masked. The vast majority of the array references took between 2 and 3 microseconds with an average time of 2.6 microseconds. This seems like a reasonable estimate for an array reference[3]. However, 0.016% of the observations were outlyers, overflowing the main mode. These did not significantly affect the average time, but the histogram is definitely bi-modal[4] with a second, very small, hump at about 257 microseconds. What is the second mode measuring?

All of the array references in the 50,000 iterations were identical, so it is unlikely that this is some uncommonly occuring branch in the normal course of an array reference. This is too long for a cache miss, and too short for a page fault. There must be some unmaskable interrupt occuring occasionally, interfering with the behavior and measurement of the array reference.

Assume the start of the array-reference is uniformly distributed with respect to the starting time of the interrupting event. A single `aref` (the machine instruction for an array-reference) takes, on average, 2.62 μ-secs. The odds of being preempted were 8 out of 50,000. So, 2.6207593 μ-secs is 8/50000 of the inter-event interval, or the average inter-event interval is $131037.966/8 = 16379.458$ μ-seconds. Since the average time of the interrupt was 236.75 μseconds, we can conclude that the interrupt occurs on average approximately once every 16616.458 micro-seconds.

There is no reason to believe that this hypothesized interrupt occurs with a regular period, but 16.6 milliseconds is remarkably close to 60Hz. We can guess that we have indirectly measured the period (60 hz) and duration (approximately 1/4 millisecond) of the regularly occuring "sequence break"[5]. This is useful information in understanding the behavior of our machine, and in interpreting subsequent metering results that have a similar distortion due to an unmaskable sequence break.

2.1.2 RPC Times

The first two columns of Table 2 presents the results of measuring the performance of one RPC call optimization in the Cache Kernel[3]. In the normal, unoptimized, case, when RPC data arrives, the arrival is `signal`led,

[2] Although it's not immediately relevant, the machine was a Symbolics 3640 running Release 6 of system software. The metering occurred in 1985. These results seem to strike the right balance between simplicity (explainable in less than a page) and interest (discernable interference in measurements), which explains the (relative) obscurity, (relative) age, and (relative) triviality, of the example.

[3] 10 years ago machines were slower, and in this instance the hardware array reference was also performing bounds checking, etc.

[4] Actually, it is probably tri-modal, in some limited sense. The grouping of data points at 171 and 172 makes it seem possible that a third hump would show up there. With this limited number of observations, though, it is impossible to tell.

[5] The name given, for historical reasons, to a periodic, unmaskable, interrupt on Symbolics' Lisp Machines.

	Round Trip RPC	Void RPC	Void RPC (fixed)
Optimized	123	55	65
Non-optimized	150	85	93
Non-optimized (with context switch)	246	167	190

Table 2: Measurement of RPC times with various optimizations

and the signal handler wakes up the process for whom the RPC call was intended. An optimization allows some RPCs to be handled directly in the signal handler.

The columns in Table 2 show the times for two types of RPCs. Round Trip RPC measures the time of an RPC that returns a value. Void RPC measures the time of an RPC that returns no value — i.e. it doesn't return at all.

The "Optimized" row in the table reports the time (from the caller's point of view) of an RPC that is handled directly in the signal handler on the receiving processor. The "non-optimized" row reports the time when the process handling the RPC is the currently running process on the receiver (so the only extra overhead is the return from the RPC signal handler). The final row reports the time when the process handling the RPC is not currently running on the receiver, but wakes up, preempts and runs when the signal handler dispatches the RPC to it.

The excess overhead of the context-switch case compared to the simple non-optimized case is identical in the Void RPC and the Round Trip RPC. Therefore the difference in the timings should be the same.

But it isn't: $246 - 150 = 96$, but $167 - 85 = 82$. (This was not noticed initially.)

Apparently, $\frac{181-167}{181-85} = 14.6\%$ of the RPCs arrived while the signal handler was still servicing the last one, and therefore required no context switch. The 167 microsecond figure was the average of approximately 85% context switches (181 usecs), and 15% looping in the exception handler and the RPC process (85 usecs).

This conclusion could have been reached purely from the average data. It is easy to miss, though (even if the investigator were aware that (246 - 150) should have equalled (167 - 85)). If the values had been recorded in a histogram, however, the bimodal nature of the data would have been *immediately* obvious to the most cursory examination.

2.2 Barriers to Histogram Use

Earlier, we claimed (based on informal sampling and anecdotal information), that the most common forms of metering compressed all observations into a small number of statistics (number of observations, sample mean, standard deviation, min/max, etc.) losing critical information about the distribution of observations.

This deviation from recommended textbook technique (see, for example, [10] or [6]), when not done out of carelessness or lack of knowledge, is driven by three considerations: (1) absolutely no prior knowledge of the distribution, (2) programmer convenience,and (3) computational overhead.

It is clear that *if* we knew the type of distribution in advance, we could more easily and efficiently completely recover the Cumulative Distribution Function (CDF). For example, in the case of a known Normal distribution we can completely recover the distribution by simply computing the mean and standard deviation. However, in large scale metering we usually have no prior knowledge of the distribution and either can not, or will not, meter again. When we have no knowledge of the distribution then we must resort to nonparametric methods; something very much like a histogram will be needed to tell us about the distribution of values.

Syntactic convenience for the programmer does not seem to be much of an issue anymore. Although programmers can toss in the computation of an average very easily, object oriented technology (C++, for example) allows one to use histograms just as easily (see Table 3 for an example), assuming appropriate histogram packages are provided in a library.

Unless the use of a histogram is this simple, it will not gain acceptance in large scale metering. Two elements are required, in combination, to provide this necessary ease of use. Object oriented technology provides syntactic convenience: the ability to transparently change the type (for example different types of histograms, or even `average` or `standardDeviation` of the data collector (by changing the type declaration and initialization of `h`) without changing any code in the body of the program. It is hard to argue that it is

`int    count;`
`double total;`
`...`
 	
`count++;`
`total += datum;`
	`EquiProbableHistogram   h(NBUCKETS);`
	`...`
	
	`h->newDataPoint(datum);`

Table 3: Code for computing average compared to recording data in a histogram.

any more difficult to add a new data point to a histogram than to add it into an `average`.

The second reqired element is semantic convenience — and object oriented programming styles do not help us there. It is important that there be no complicated[6] parameters at initialization time: no bucket size, or low or high bounds. A programmer needs no prior knowledge to compute an average. A histogram must be as easily deployable. This can be accomplished if the histogram is *self scaling* – i.e. the histogram gradually chooses its own bounds and bucket sizes in response to the data that it sees.

A further hurdle is computational overhead: one of the strengths of using conventional histograms (assuming bucket size and bounds are known) is that it is only marginally more expensive than computing (for example) the standard deviation. For each new observation, you simply subtract the low bound, divide by the bucket size, indirect and increment. If the semantic convenience of self-scaling histograms comes at too high a price in computational overhead, then they are not useful. Programmers will not use self-scaling histograms for quick and dirty metering, much less insert them as permanent fixtures in kernels or long running programs/servers. So, our goal must be *efficient* self-scaling histograms.

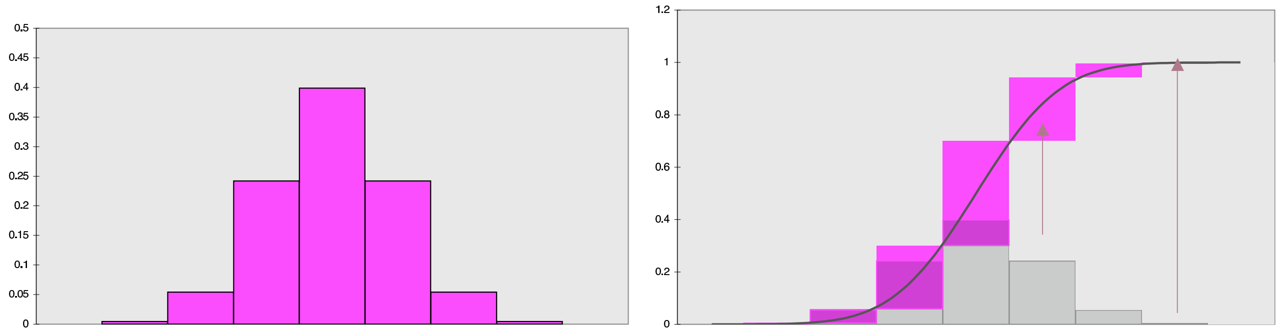


Figure 2: The relationship between a histogram and the CDF.

2.3 Reconstructing the Distribution Without Storing Observations

In the context of performance measurement, histograms are a means to an end: recovering the *entire* distribution as much as practical. We can describe the probability distribution of a random variable by the CDF.

2.3.1 Relationship between histograms and CDF

Figure 2 graphically shows the relationship between a histogram and the corresponding CDF. The figure on the left represents an equi-valued histogram (a histogram in which every bucket is the same width) of measurements of an event whose duration has a normal distribution. The figure on the right, scaled appropriately, shows the connection between the histogram and the CDF. Starting from the left (the bucket with the lowest range of values), each bucket is shifted upwards, so that its bottom coincides with the top of the bucket to its left. The CDF enters each bucket at the lower left corner, and exits at the upper right corner.

Given that histograms are a means to an end, it follows that not all histograms are created equal. Different histograms can capture the same CDF. For example, equi-probable (or "equi-depth") histograms are an alternative to conventional histograms. Conventional histograms, called equi-width or equi-valued histograms,

[6]It is not so much the "complexity" of the parameter values as the fact that if you choose them badly, you lose significant information.

choose fixed width buckets — if the bucket width is b, and the minimum value is x_0, then the ith bucket counts the occurences of values in the interval $[x_0 + ib, x_0 + (i + 1)b)$. Equi-probable histograms, on the other hand, choose bucket boundaries such that the number of events falling into each bucket is equal. So a 10 bucket equi-probable histogram describing a sample with 10,000 observations, chooses the bucket boundaries such that 1,000 observations fall into each bucket.

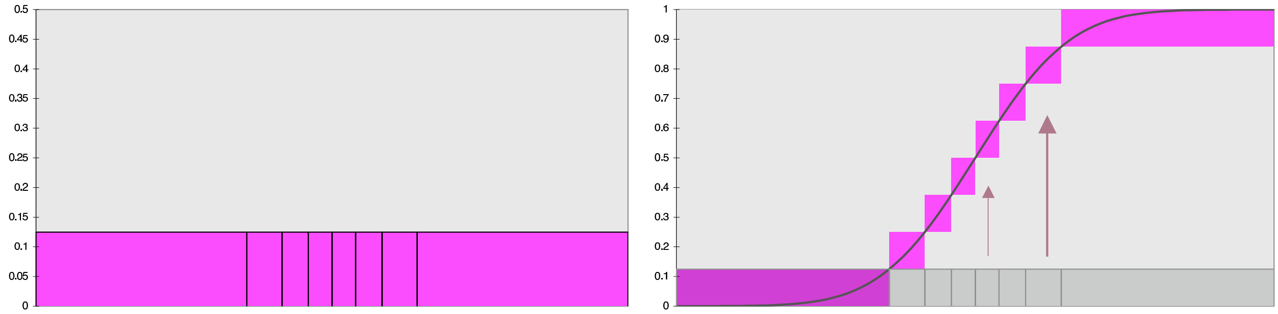


Figure 3: Converting an equi-probable histogram to a CDF.

Figure 3 show how an equi-probable histogram of a normal distribution is converted to the same CDF as before, albeit with different bucket boundaries. The figure on the left shows an equi-probable histogram (note that all buckets are the same height, but most have different widths). The figure on the right shows how the same procedure of shifting buckets aligns the bucket boundaries with the CDF.

It is also possible to convert histograms to a pmf (probability mass function) or a pdf (probability density function). We can divide the height of a bucket by its width, in order to get an approximation to the relative densities of each bucket.

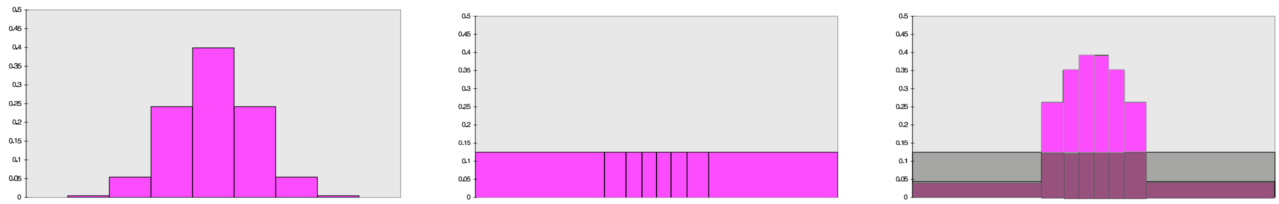


Figure 4: Converting a histogram to a pmf.

Figure 4 shows this process for both equi-width and equi-probable histograms. All buckets in an equi-width histogram have the same width, so the histogram, itself, is the pdf. Equi-probable histograms (as can be seen in the center figure) have equal-heights, but different widths. The rightmost figure shows how the equi-probable histogram is converted to a pdf by dividing each bucket by its width. However, interpreting quantized pdf's is sometimes confusing, while CDF's are defined for all distributions, continuous or discrete, and have a natural interpretation. Consequently, we restrict ourselves to CDFs. For purposes of illustration only, we occasionally produce pdf's when the graphical representation is more familiar and intuitive.

We have spoken so far of both conventional (equi-width) and equi-probable (equi-depth) histograms. More generally, we can approximate the CDF by computing *any* B points along the distribution. The B points can be used to interpolate a monotonically increasing function passing through these points (say, for example, by constructing a cubic spline using these as control points). We can now choose any other B points along this CDF as new bucket boundaries, to create a new histogram that approximates the same CDF.

Figure 5 shows how this technique can be used to adjust the bucket values in an equi-width histogram when a new observation arrives that is lower than the current minimum. We hold the number of buckets fixed, and compute new bucket boundaries with a new minimum and a larger width. Alternatively, we can use this technique to convert an equi-width histogram to an equi-depth histogram — or to *any* histogram we choose.

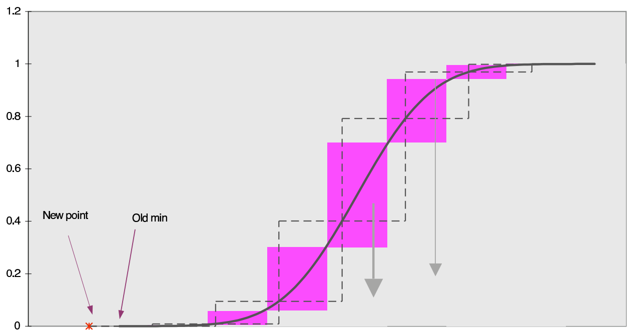


Figure 5: Shifting points on the CDF back down into buckets for to construct a new histogram. In this case, a new observation has changed the minimum value and increased the range of values covered by this equi-width histogram, and consequently has changed the bucket size.

2.3.2 Accurate approximations of the CDF using histograms

But our real goal is to choose a histogram that gives us the most accurate approximation to the underlying CDF. So we must answer the question: "What choice of points will minimize the error between the interpolated CDF and the real CDF?"

It is quite possible for two substantially different distributions to coincide at all or most of the B points you happen to choose as quantiles. (Consider Figures 6 and 7. The discrepancy shows up more clearly in the pdfs.)

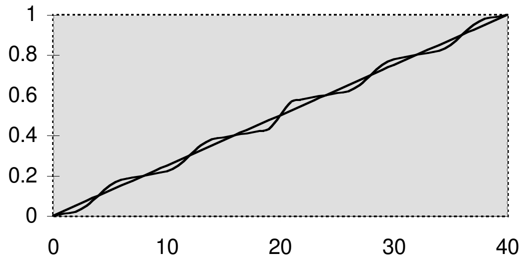


Figure 6: Two CDFs of different distributions, both having identical quantiles at multiples of .1.

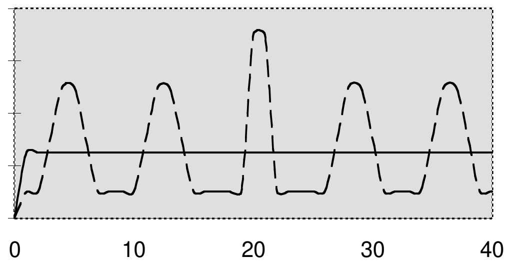


Figure 7: pdf's corresponding to CDFs in Figure 6

Recall that we have no prior knowledge of the distribution. Given B points known to be in the function that we are trying to measure, all monotonically increasing functions are equally likely, subject only to the constraint that they pass through those B points. So one way of viewing our problem is to choose the points that most tightly constrain the number of other possible functions that might have generated the same B points.

We can quantify the *available error* between two points, (x_i, y_i) and (x_{i+1}, y_{i+1}). All possible monotonically increasing functions that pass through those two points must be bounded below by the step function going from y_i to y_{i+1} at x_{i+1}, and bounded above by the step function going from y_i to y_{i+1} at x_i, as depicted in Figure 8. We define *available error* between those two points to be $(x_{i+1} - x_i) \times (y_{i+1} - y_i)$. This rectangle[7] represents the area that can be covered by other functions passing through these two points. For a given function, the sum of the available error for each of the B points is a good predictor of the expected error between the interpolated function and the actual underlying function. A good heuristic, then, is to choose our B points to minimize the available error.

[7]This measure is an oversimplification. It assumes that all points are equally likely to be in a legal function passing through our points. However, since we are only considering monotonically increasing functions, the probability that a given point (x, y) will be in some function is proportional to $((x - x_i) \times (y - y_i)) + ((x_{i+1} - x) \times (y_{i+1} - y_i))$. The area of the rectangle is a reasonable approximation, though.

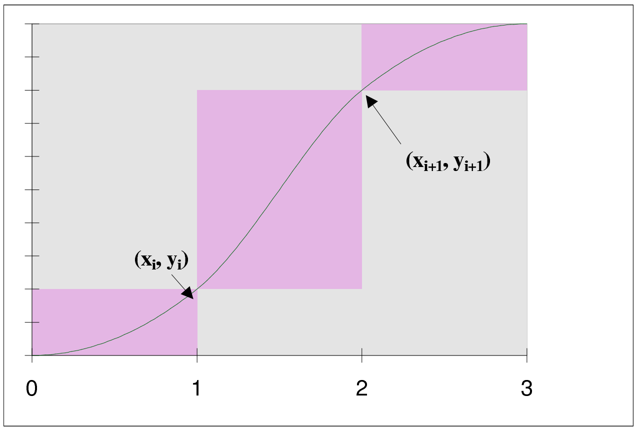


Figure 8: The available error between (x_i, y_i) and (x_{i+1}, y_{i+1}) is depicted by the shaded rectangle in the center of the figure.

The obvious first approach is to space the points out "equally" along the function. In this context, we can mean several different things by "equal" — the two most obvious being equally spaced along the percentile axis, or equally spaced on the value axis. These two meanings correspond to two different types of histograms: histograms with equi-probable cells and histograms with equal-sized cells[8].

Received wisdom is that equi-probable histograms will be more accurate than equi-sized histograms. Computing the available error for each, however, casts some doubt on this truism.

The range (pctile axis) of each function will be the interval[9] $(0, 1)$. The values in the domain will range from some minimum to a maximum real value, call this range R =max - min. First, consider using equi-probable cells. If there are B buckets (recording $B + 1$ points), then each bucket is $1/B$ tall (equally likely) and the widths must sum to R (to cover all possible observations in the domain). Thus, the available error for the entire function will be R/B. Now consider equi-sized cells: each cell will be R/B wide (the range of values is divided into B equi-sized buckets), and the heights must sum to 1. Therefore, the available error will also be R/B.

So, the total available error for *any* function that represents the CDF using equi-probable cells will be identically equal to the available error of that function using equi-sized cells. Further, the available error is independent of the function: dependent only upon its range. For some *particular* functions, equi-sized cells will be more accurate; for other *particular* functions, equi-probable cells will be more accurate. But it is impossible to know in advance.

R/B is obviously an improvement over the worst case available error. One can (badly) choose the points such that a single cell goes from $(\min, 0)$ to a point arbitrarily close to $(\max, 1.0)$, which would cause the total available error to approach R. So a B bucket equi-probable or equi-sized histogram reduces the available error by a factor of B.

But how bad is R/B compared to the *best* we can do?

Intuitively, equi-probable buckets seem like they present problems. Consider an operation that takes roughly 20 microseconds to execute. Imagine that it page faults 0.1% of the time, and that each page fault adds about 18 milliseconds to the execution cost. An equi-probable histogram with as many as 50 buckets still won't really show the page fault behavior because the page faults correspond to only 0.1% of the observations. However, the page faults contribute roughly 50% to the sample mean. The available error in the last cell is overwhelmingly large.

An equi-sized histogram will accurately capture the page fault behavior; they will be isolated from the

[8]These two types of histograms are the easiest for most people to interpret, and are what we are most comfortable with. (In fact, if you use the term "histogram" with no qualification, most people would assume you are referring to histograms with fixed, equal-size, buckets).

[9]Open interval for unbounded functions, closed interval for bounded functions. In practice, for functions that are unbounded, we will never see the real "min" or "max" (which, are, after all, unbounded), but the more observations we make, the larger the range between the sample min and max.

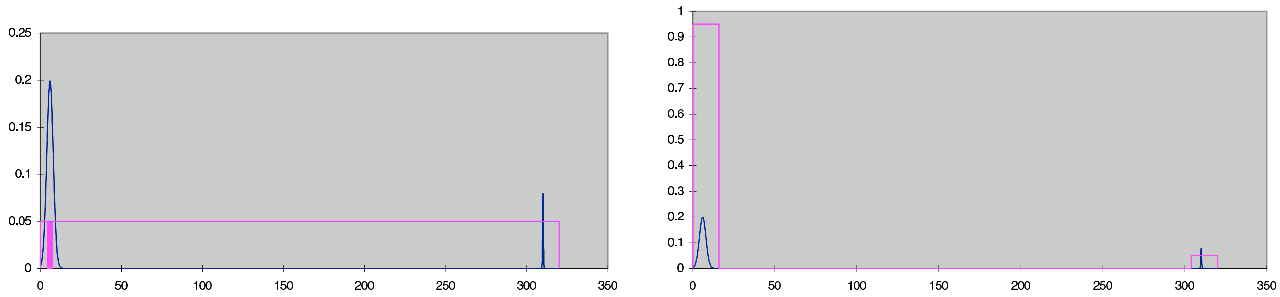


Figure 9: Information loss in equi-probable and equi-width histograms.

main body of the distribution and will likely provide an exact count of the page faults. However, them entire main mode of the distribution (99.9% of the data points) will fall into a single bucket, obscuring all structure: is the main mode normally distributed? Exponentially distributed? We don't know, because the presence of even a single page fault or other anomolous outlyer would drown the entire main mode. Figure 9 represents this graphically in a slightly more easily displayed case with a 20 bucket histogram, and a smaller distance between the modes.

We'd like a histogram that would avoid this kind of information loss — the solution is presumably a histogram with a total available error of significantly less than R/B.

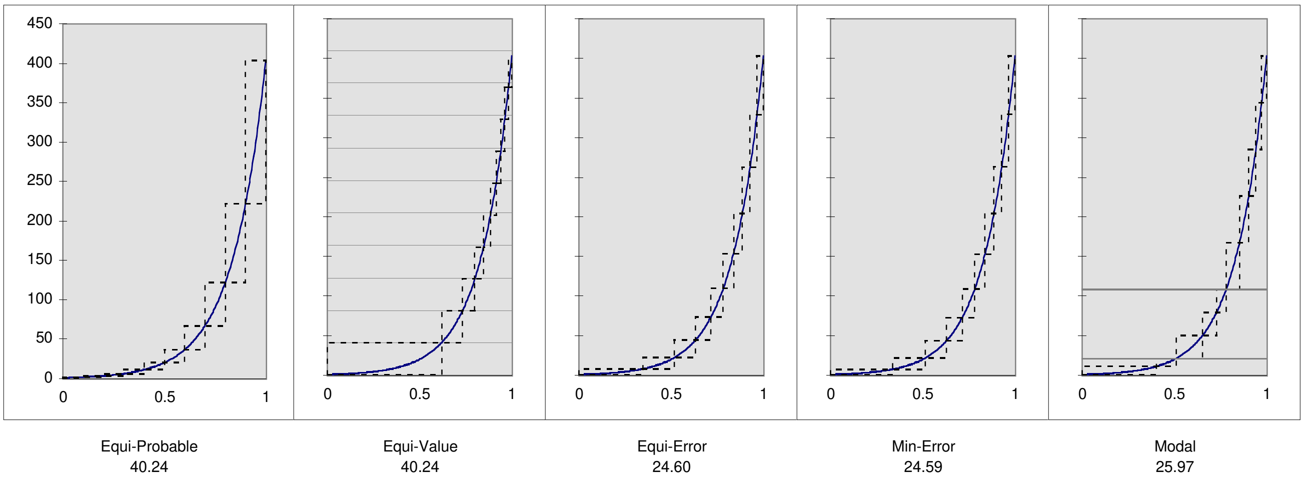


Figure 10: *Available error* for exponential distribution. $\mathrm{CDF}^{-1} = e^{6p}$ for different algorithms used to choose observation points. Here $B = 10$, and we look at 11 points.

Figure 10 shows, for one example function (e^{6p}), how different choices of points in 10 bucket histograms, accumulate different amounts of available error. We can see that the equi-sized cells accumulate larger errors where many observations fall into a single cell (which makes sense, because we have lost more information about the curve). This error in equi-sized histograms is balanced by the error in cells in equi-probable histograms where the range of values is large. The total available error in both types of histograms are equal, and approximately 40.24. The placement of the 11 points that minimizes the total available error yields a total error of approximately 24.59, a little over one half as much error. Table 4 shows the available error for some other functions: in a LogNormal(5.0,1.0) distribution R/B is 5 times worse than the minimum, while in a Normal(5.0,1.0) it is only about 1.5 times worse than ideal.

In general, the more the function deviates from linear, the worse R/B is, compared to the minimum. Since the gap between R/B and the minimum possible available error can be large, we developed algorithms that try to do a better job at minimizing available error. In practice, these turned out to be substantially more accurate than equiprobable histograms.

					Normalized			
	Equi-width				EquiWidth			
Function	EquiDepth	EquiError	MinError	Modal	EquiDepth	EquiError	MinError	Modal
e^{6p}	40.24	24.60	24.59	25.97	0.1	0.061	0.061	0.065
Normal(5.0, 1.0)	0.743	0.529	0.526	0.553	0.1	0.071	0.071	0.074
LogNormal(5.0, 1.0)	612.00	123.08	122.76	147.09	0.1	0.020	0.020	0.024

Table 4: *Available error* for different histogram types, using 10 buckets, for some example functions. For unbounded functions the domain is $[10^{-4}, 1-10^{-4}]$. The top row corresponds to the graphs in Figure 10.

3 Algorithms for Self-Scaling Histograms

Any algorithm for self-scaling histograms must solve several problems. First, where do the bucket boundaries belong? For a B bucket equi-sized histogram the answer is obvious: B equally spaced buckets between the min and max value seen so far. For equi-probable, or for our hypothetical min-error, histograms, the answer is not straightforward. Even for histograms with equal sized buckets, a second problem crops up: if the minimum or maximum change, then we must change the bucket boundaries. How do we re-apportion the observations that have fallen into the existing buckets?

3.1 Basic Algorithms

3.1.1 The P^2 algorithm for EquiProbable Histograms

Jain et. al. [9] provided one elegant approach to this problem. The P^2 algorithm dynamically calculates quantiles using fixed (and very small) storage. That paper briefly discussed an extension of the P^2 algorithm to histograms with equi-probable buckets, rather than equi-sized buckets.

The approach follows from the definition of a quantile: if a number $Q(p)$ is the p-quantile, then any observation will be less than $Q(p)$ with probability p. As N counts the number of observations we've seen, we successively guess a $Q_N(p_i)$ as an approximation of $Q(p_i)$. For a B bucket histogram, sort the first $B+1$ observations, and assign them, in order, to $Q_B(p_i)$, where $p_i = i/B$. For each observation, D, we find the bucket, i, such that $Q_N(p_i) \leq D < Q_N(p_{i+1})$. We increment the count of observations associated with each bucket greater than i. If other than Np_i out of the N observations in the sample fall below $Q_N(p_i)$, we raise or lower the next approximation, $Q_{N+1}(p_i)$, until observations fall below $Q_N(p_i)$ with probability p_i. The P^2 algorithm assumes that the inverse cumulative distribution function can be locally approximated by a quadratic function. If k out of N observations fall below $Q_N(p_i)$, and $k \neq Np_i$, then what we thought of as $Q_N(p_i)$ is really $Q_N(k/N)$. If $|p_i N - k| \geq 1$ after N observations, then we adjust $Q_{N+1}(p_i)$ to be the point on the quadratic curve going through $Q_N(p_{i-1})$, $Q_N(k/N)$, and $Q_N(p_{i+1})$. We perform the same adjustments on all p_i. See [9] for more details[10]. As $N \to \infty$, $Q_N(p_i)$ converges to $Q(p_i)$ as rapidly as the sample statistics do.

3.1.2 Reconstructing the entire distribution

Although, [9] does not specify how to recover the complete distribution, we define the operation `getQuantile(q)` on all histograms to return (an approximation of) the q quantile. Consider Figure 11. The P^2 algorithm utilizes a quadratic function $Q_1(q)$ going through points 0, 1, and 2 to adjust the quantile at 1, and a quadratic function $Q_2(q)$ going through points 1, 2, and 3 to adjust the quantile at 2. How do we compute `getQuantile(q)` for q between points 1 and 2? We use a weighted average of Q_1 and Q_2: $\frac{x_2-x}{x_2-x_1}Q_1(x) + \frac{x-x_1}{x_2-x_1}Q_2(x)$. Since Q_1 and Q_2 are 2nd degree equations, and we multiply by x, the resulting approximation is a cubic spline. The closer we are to x_1 the closer `getQuantile` is to Q_1, closer to x_2 the more Q_2 is weighted.

It is well known that you can construct a 2nd degree function going through any three points. However, if we require the function to be monotonically increasing, then the set of valid triples is limited. Consider three points, $(X_0, Y_0), (X_i, Y_i)$, and (X_1, Y_1). If the quadratic function going through all three is monotonically increasing then, in the local region $[X_0, X_1]$, (X_0, Y_0) must be the min and (X_1, Y_1) must be the max. We can easily prove that if (X_i, Y_i) lies below the function LB(x) then there exists a minimum (a point (x, y) with $y < Y_0$ and $X_0 < x < X_i$. Similarly, if (X_i, Y_i) lies above UB(x) then there exists a maximum (x, y) such that

[10] But note that there is a typo in step B.3 of Box 2. $(i-1)(n-1)/b$ should be $(i-1)(j-1)/b$

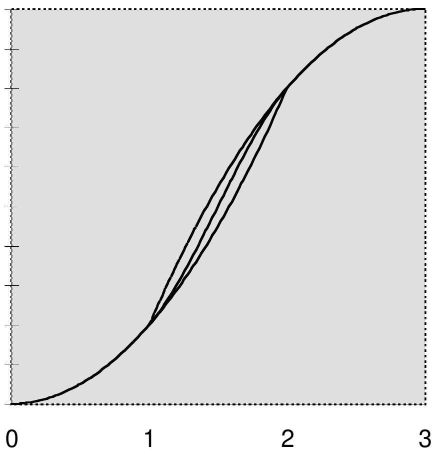
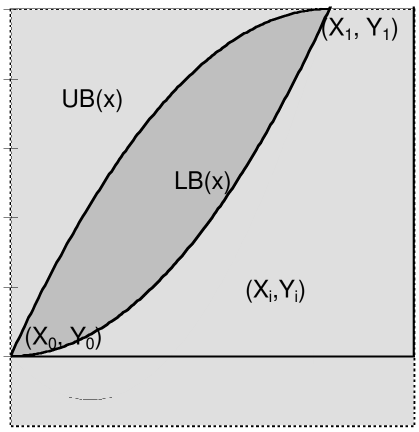


Figure 11: We compute quadratic function $Q_1(x)$ going through points 0, 1, and 2 (see Figure 13). We compute quadratic function $Q_2(x)$ going through the points 1, 2, and 3. `getQuantile(x)`, for x between 1 and 2, returns a weighted mixture of Q_1 and Q_2, with the weights proportional to the distance from points 1 and 2.



$$LB(x) = \frac{Y_1 - Y_0}{(X_1 - X_0)^2}(x - X_0)^2 + Y_0$$

$$UB(x) = Y_1 - \frac{Y_1 - Y_0}{(X_1 - X_0)^2}(x - X_1)^2$$

Figure 12: All monotonically increasing quadratic functions that contain both (X_0, Y_0) and (X_1, Y_1), must lie between LB(x) and UB(x). (X_i, Y_i) cannot be a point on such a function. Either the function is not quadratic, or it is not monotonically increasing in this interval.

$y > Y_1$ and $X_0 < x < X_i$. (see Figure 12 on page 11 for a graphical illustration and definitions of LB(x) and UB(x)).

We resort to linear interpolation if (X_i, Y_i) cannot be part of a monotonically increasing quadratic function, i.e. it lies outside of LB(x) and UB(x).

3.1.3 Equi-size Histograms

The calculation of available error implies that histograms with equi-sized cells should have error comparable to equi-probable histograms. However, in contrast to equi-probable histograms, we no longer have to search for bucket i to record each observation. We simply increment the count in bucket $\left\lfloor \frac{D - \min}{\text{bucketsize}} \right\rfloor$. This cost is independent of the number of buckets. Nor do we have to adjust the values that border each bucket. So, if the error is really equivalent to equi-probable histograms, we would always choose equi-sized histograms, since the overhead is so much lower.

The buckets in the equi-size histogram remain constant unless the min or max changes. If a datum D falls outside of $[\min, \max]$, then we must adjust all values.

We update min and max and use these to update bucketsize to $\frac{\max - \min}{B-1}$. To compute the counts in each of the new buckets we again assume that the CDF is locally quadratic. For the new jth bucket, we compute the pctile of $\min + j * \text{bucketsize}$ and $\min + (j+1) * \text{bucketsize}$ based on the counts in the old buckets. We compute each pctile by the method described in Section 3.1.2[11]. The new count for bucket j is the number of observations seen so far times the difference between the pctiles .

As the number of buckets goes up, equi-sized histograms are substantially faster at storing data than equi-probable histograms. See Figure 17.

3.1.4 Min-Error and Equi-Error Histograms

We have noted that the available error for equi-probable and equi-sized histograms could be much higher than the minimum available error. We will construct a histogram that dynamically adjusts its buckets to minimize the available error for the entire distribution. We note that since we are no longer using buckets of equal size or equal probability, we now are required to store two values per bucket boundary. (Measurements described

[11]However, here we approximate the CDF, not the inverse CDF, since we are going from values to pctiles. From here on, we are going to refer to both the CDF and the inverse CDF as the CDF and expect the reader to distinguish from context.

$$
\begin{aligned}
Q(x) &= Ax^2 + Bx + C \\
A &= \frac{X_2Y_1 - X_1Y_2}{X_1^2X_2 - X_1X_2^2} \\
B &= \frac{X_1^2Y_2 - X_2^2Y_1}{X_1^2X_2 - X_1X_2^2} \\
C &= 0
\end{aligned}
$$

Figure 13: Q(x) is the local quadratic approximation going through the 3 points. Translate coordinates (X_0, Y_0) to $(0, 0)$ (and $X_i = X_i - X_0$), to simplify.

$$
\begin{aligned}
Ax^2 + \frac{2B - AX_2}{3}x - \frac{BX_2 + Y_2}{6} &= 0 \qquad (1) \\
Ax^2 + \left(B + \frac{Y_2}{X_2}\right)x - Y_2 &= 0 \qquad (2)
\end{aligned}
$$

Figure 14: Solve equation 1 for x to minimize the total available error, or equation 2 for x to equalize the available error in each bucket.

in Figure 16 show that the error is still lower than equi-sized or equi-probable histograms with twice as many buckets).

The *Min-Error Histogram* records data in the same way as an equi-probable histogram. Upon observing a new datum, the appropriate bucket is found, and the index is incremented. This changes the distribution function, and we must determine whether the bucket boundaries need to be moved to minimize available error.

We do not have a closed form for the entire distribution, but we have a piecewise quadratic approximation. There is a point, (X_i, Y_i) on the distribution function between each pair of buckets. For a given point we can assume that its neighbors are fixed. For the quadratic approximation $Q(x)$ on $[X_0, X_2)$, we want to choose an $(x, Q(x))$ such that the sum, $S = (x - X_0)(Q(x) - Q(X_0)) + (X_2 - x)(Q(X_2) - Q(x))$, of the available error for the two buckets $[X_0, x)$ and $[x, X_2)$ is minimized. $Q(x)$ is a quadratic with known coefficients (in terms of the existing points X_0, X_1, and X_2), so we take the derivative of this cubic equation and solve for x. (See Equation 1 in Figure 14.) Note that $S = (X_2 - X_0)(Q(X_2) - Q(X_0))$ for both $x = X_0$ and $x = X_2$, and this value is a local maximum. So there will be a unique root of the derivative inside the interval and that root will be the minimum value of the available error. We set the new X_1 to x.

If X_0 and X_2 are both held fixed, then this computation does minimize the available error. However, a global minimum for the entire distribution can only be computed by simultaneously recomputing *all* the buckets. By deriving each $Q(x)$ from existing points, but solving for each new X_i and minimizing the sum using the *new* X_i's, we can set up a system of non-linear equations. We can solve this system and determine the ideal placement of all buckets, and recompute their counts. This is prohibitively expensive, so we choose a local approximation.

In practice, we only recompute the bucket boundaries of "dirty" buckets. A bucket b is dirty if a new data point has been added to b, or if its immediate neighbors have been adjusted. When a bucket's boundary is adjusted we clear the bucket's dirty bit. We limit unnecessary recomputation in one further way. If $|X_1 - x| < 1/N$ after N observations, then we don't adjust X_1 (it is "close enough").

In practice, although Min-Error histograms are far more accurate than either equi-probable or equi-width histograms, they do *not* perform as well as they could. Recall that $(X_i, Q(X_i))$ is not a point that is necessarily actually on the CDF. It is an approximation that converges to a point on the CDF after sufficient observations. If $Q(X_i)$ is not held fixed for long enough, then the approximation of X_i will not necessarily be accurate. In spite of the attempt to dampen the recomputations, pathological arrival order of the observations can sometimes cause the Min-Error algorithm's estimate of the $Q(X_i)$'s to jump around enough so that convergence to the CDF is slowed down, increasing the measured error. Further, computational costs are high from recomputing bucket boundaries.

We can use a more stable placement of cells in order to speed convergence to the CDF, yet still place the cells in a way that reduces the available error.

A good heuristic to choose points from the CDF is to attempt to make the available error for each bucket equal to all others[12]. This avoids creating large cells with lots of errors, and avoids wasting data points by placing them too close together — all cells are weighted equally in the error. In practice, as predicted, these *Equi-Error* Histograms appear to yield close to the minimal available error (see Figure 10 and Table 4).

[12] One can prove that, in the limit, as the number of buckets increases, this heuristic will strictly minimize total available error. The fewer the buckets, though, the more it differs from the ideal Minimum Error choice of points.

The algorithm is very similar to Min-Error. Again, we only make local adjustments. However, instead of positioning a bucket boundary to minimize the sum of the available error in the adjacent buckets, we try to *equalize* the available error. (See Equation 2 in Figure 14.) Although equalizing errors seems to naturally place cells in more stable positions, we further dampen the recomputations of bucket boundaries. After computing a new candidate X_i that equalizes the error of two buckets, we check to see that the new placement also reduces the sum of the available error. (We don't try to minimize it, but we don't adjust the bucket unless it reduces the total error). Using this algorithm, the points are stable, converge rapidly to the CDF, and the measured error of the entire reconstructed CDF can be orders of magnitude less than for equi-probable or equi-sized histograms (Figure 16).

3.2 Modal Histograms

Equi-Error histograms yield very small errors, but require twice the storage for a given number of buckets than Equi-size histograms. Further, the cost of inserting a new data point into an equi-error histogram is proportional to the number of buckets, while the insertion cost into an Equi-Sized Histogram is roughly constant.

Can we construct a histogram with storage and computational overhead of the equi-sized histograms, but with the accuracy of the equi-error histograms?

We can come close by using *Modal Histograms*. (See Figure 10 and Table 4). Modal Histograms divide the distribution into modes, and each mode is treated like an equi-sized histogram, with its own bounds and bucket size. When a new observation arrives for a modal histogram, we first determine which mode it belongs in. (The computational cost of this lookup is a function of the number of modes in the histogram, and independent of the number of buckets.) Then, we subtract the lower bound of that mode, divide by the bucketsize, and increment the appropriate bucket. (As an optimization, as long as the number of observations within a mode are fewer than the number of buckets, we store a list of the individual observations and defer computing the bucket size.) This is a small number of operations, and independent of the number of buckets. If a data-point does not fit within any of the current modes, we use this as an opportunity for reselecting the modes, and reallocating the data in each bucket to the new modes.

Modal histograms store a lower bound, a bucketsize, and a bucket count for each mode. For a fixed number of modes this overhead is independent of the number of buckets.

We have empirically determined that three modes seems like a good number in most cases of interest. I initially guessed at tri-modal histograms based on an intuitive correspondence with a main mode, an underflow and an overflow in the data (the nomenclature remains in my code). Three *does* empirically seem to be a good number of modes in the histogram — however, not for the reasons I envisioned.

For roughly symmetric, unimodal distributions of data, tri-modal histograms[13] only offer an incremental reduction in error over bimodal-histograms. In practice, the distributions of values that I've encountered most commonly appear to be bimodal distributions or unimodal distributions with long tails. In the bimodal distribution case, a third histogram mode is needed to reduce error in the sparsely populated area between the modes in the distribution. In the case of long-tailed distributions, the density of the values in the tail is often uneven. While, generally, two modes of the histogram describe the mode in the data with very little error and one histogram mode is sufficient to describe the tail, occasionally, four-moded-histograms reduce the error, noticeably, over tri-modal histograms[14]. The algorithms described here do not depend in any way on the precise number of modes — the number can be extended to four or more, although I have not yet found the need to do so.

We can prove that, for a B bucket modal histogram with 3 modes, we can minimize the total available error by allocating buckets to modes using the following procedure. Compute the available error for each mode as if it were a single bucket. We label the available error of the three modes E_1, E_2, and E_3. Let $E = \sqrt{E_1} + \sqrt{E_2} + \sqrt{E_3}$. We allocate $B_i = \frac{\sqrt{E_i}}{E} B$ buckets to mode i. The total available error for the B bucket histogram is $\frac{E^2}{B}$. (Note that B_i might not be an integer, in which case you must choose the best conversion to integers.)

[13] I am discussing the number of "modes" in the distribution of the data, and the number of modes in the histogram. The former is a function of the data you observe. The latter is a fixed integer describing the structure of a data-structure.

[14] Alternatively, for problematic distributions one can increase the number of buckets. The error reduction here is not by reducing the available error, but rather it permits the end of the tail to be stored as discrete values, with 0 error.

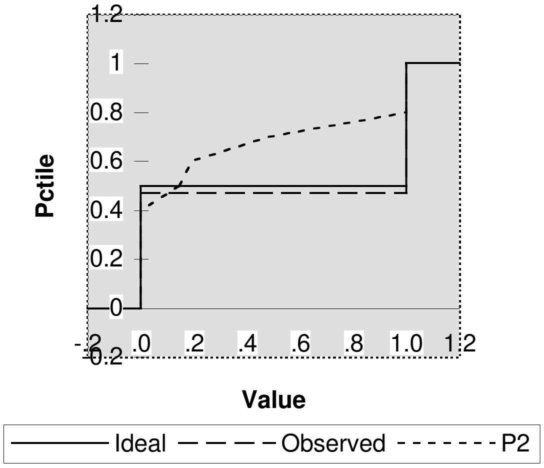


Figure 15: Ideal CDF of Bernoulli Process, compared to CDF's constructed from the complete list of observations and from a 10 bucket P^2 histogram.

When we must divide the modal histogram up into modes, we choose 2 points, m_1 and m_2, on the CDF as boundaries between the modes. We choose them such that E is minimized. For efficiency, rather than choosing the best points *anywhere* on all of the local $Q(x)$'s, we simply inspect points that are currently on the border between buckets.

4 Discrete Distributions

As noted in [7] about the P^2 algorithm, the algorithms described up to this point can give incorrect values for discrete distributions, or for distributions that are discontinuous near a quantile (see Figure 15).

Consider random observations selected from a Bernoulli process. The ideal CDF is a step function that goes from 0% to 50% at 0.0, and then steps to 100% at 1.0. The only observations that we should see are either 0.0 or 1.0 in roughly equal numbers. The P^2 algorithm produces a histogram, however, that may claim that 40% of the observations are greater than 0 and less than 1, or, more narrowly, that 10% of the observations lie between 0.1450 and 0.1937 (see Figure 15), when none of the observations actually did.

Discrete distributions are common. Clocks return integer values — often only multiples of some clock granularity. Integers are more common data-types than floating point numbers. The algorithms, as presented above, only deal well with continuous distributions. We would like our algorithms to recover the CDF more accurately in cases of discrete distributions. We discuss the modifications to our algorithms in the context of P^2 for simplicity, but the modifications can be (and are!) applied equally well to equi-error histograms.

A simple first pass at a solution (as suggested in [7] and alluded to in [9]) is to count the frequency of individual values as long as practical. If the number of distinct values exceeds the number of buckets, we convert the histogram to the P^2 algorithm (We refer to this algorithm as P^2_D (for P^2_{discrete})). This algorithm has the drawback that the accuracy of the histogram changes abruptly as soon as the number of distinct values exceeds the number of buckets even by 1. We'd prefer more graceful degradation; the abrupt behavior implies that choosing the wrong number of buckets (before we've seen any of the data) can have significant effect when the distribution consists of a number of discrete values close to the number of buckets. Such a penalty for choosing the wrong value for a parameter conflicts with our stated requirement for conceptual simplicity.

A more promising approach is to adapt the quadratic interpolation algorithm on a bucket-by-bucket basis, rather than converting from discrete to approximations in all buckets at once. Each individual bucket might represent discrete values *or* regions of a smooth curve. We refer to this modification of P^2 as P^2_S (for P^2 with steps).

If we flag a bucket as discrete it means that the count of hits in that bucket *all* were *exactly* the value of the start of the bucket, and that the value at the start of the next bucket was the next highest value seen.

If a data point is observed that lies within, but not at the start of, the step function, then we must decide whether to mark the bucket as continuous, or split this bucket. Splitting the bucket forces us to combine two other buckets, in order to add one new "step" bucket while maintaining a constant number of buckets. We choose to either split the bucket or mark the bucket as continuous by choosing the alternative which minimizes the total available error. Note that a bucket marked as discrete has 0 available error, since it identically reconstructs the distribution. This means that a discrete bucket is unlikely to be adjusted by a neighbor, because no modification will be made unless moving the bucket boundary (and losing the information that it is discrete) reduces the total available error. So the reduction in the neighbor's error has to be larger than the entire available error in the discrete bucket (if it were no longer discrete).

This method of flagging discrete buckets will not work for equi-sized or modal histograms. The boundaries of the buckets are predetermined based on the lower bound of the mode and the bucket size. If the discrete observations are spaced erraticaly then the cells don't correspond to single values and there's nothing to flag.

Instead, we compute the granularity of the data values we've seen by using a generalized version of the Greatest Common Denominator, extended into the reals. If a new data value is not congruent to 0 mod the current granularity, we recompute the granularity by taking GCD(granularity, datum). If the granularity decreases too frequently we decide that this is a continuous distribution and give up trying to compute the granularity.

We use the granularity when we try to describe the distribution. `getQuantile` only returns results that are multiples of the granularity. For densely populated discrete distributions this returns a step function that can identically recover the complete distribution. If the distribution is sparse, or unevenly distributed, then this can smooth out the curve within a single bucket, yielding observations that weren't actually seen — however, it is still a substantial improvement over the assumption that the distribution is continuous.

5 When You *Really* Want Equi-Probable Histograms ...

P^2 histograms are more expensive than modal or equi-error histograms, and have much larger errors. However, sometimes it is useful to generate quantile-quantile plots to compare a histogram against a known distribution. In these cases it is desirable to have as accurate as possible values *at the quantiles*, but we don't care about errors on the rest of the distribution. For this we need EquiProbable histograms. There are several ways to improve the performance of P^2_S.

We first reduce the cost of adjusting the position of the quantiles.

Assume a given histogram has B buckets. As described above, we store the value of each quantile in an array $Q[i], 0 <= i <= B$, where $Q[i]$ should be an approximation to $Q(i/B)$. $N[i]$ stores the number of observations seen which were below $Q[i]$. A given observation is less than $Q[i]$ with probability $p = i/B$, in which case we increment all $N[j], j \geq i$. If, after n observations, $N[i] \neq \mathit{Round}[pn]$, then we set $N[i]$ to $\mathit{Round}[pn]$ and adjust Q[i] by interpolation. Let $N_n[i]$ represent the value of $N[i]$ after n observations, then also with probability $p, \mathit{Round}[p(n+1)] = N_{n+1}[i] = N_n[i]+1$. If an observation falls below $Q[i]$ at the same time as $\mathit{Round}[pn]$ increases by 1 then no adjustment is necessary, and no information is lost. However, when the nth observation is less than $Q[i]$, but $N_n[i]$ hasn't yet increased by 1, $N[i]$ must be decremented and $Q[i]$ re-interpolated. Then, when $N_{n+m}[i]$ is finally $= N_n[i]+1$ we must *increment* $N[i]$, and re-interpolate $Q[i]$.

Therefore, even assuming that $Q[i]$ is at the perfect value to start with, there is a $2p(1-p)$ chance of the value being adjusted for no real reason, introducing unnecessary computational overhead and, perhaps, a small inaccuracy.

By delaying the adjustment of $N[i]$'s until after an appropriate number of observations we can both reduce overhead and increase accuracy. What's a reasonable delay?

First, recall the reason to avoid delaying. Let $Q[i]'$ be the "ideal" value of $Q[i]$, i.e. the value it would be *if* we had incremented $N[i]$ and interpolated $Q[i]$. If we then observe an observation, s, which falls between $Q[i]$ and $Q[i]'$, then by delaying we have lost some information (the incrementing starts at a different bucket than it would have if we had adjusted in a timely fashion). The odds of an observation s falling into one of those cracks is only $\frac{1}{3n}(B - \frac{1}{B})$, which decreases as $1/n$ as n grows. If we determine an acceptable probability of losing information, say p_L, then we can probably safely delay $\frac{3p_L n}{B}$ observations before adjusting. Note that loss of information simply slows down convergence, but doesn't corrupt the data. The measurements confirm that there is vanishingly small difference between P^2 with constant adjustment, and P_2 delaying with $p_L = .01$.

In the results we present, we refer to the above optimization of P^2_S as P^2_S-Fast.

By using binary search rather than linear, and by only storing the number of observations *in* a bucket, rather than the total number of observations below, we remove the requirement of touching *every* bucket for each new data point. We still need the absolute index when we adjust the quantiles, but this now happens infrequently and we are willing to linearly scan the buckets in these cases.

By delaying adjusting the quantiles, and by maintaining discrete step functions where useful, we have lost the property that there are an equal number of hits in each bucket. This means we actually have to store a count of the hits. If we bound the delay for adjusting quantiles to some number D_B, then we know that the entry count in each bucket can differ by at most D_B from the ideal number of observations in that bucket. We can then store only the delta, limiting the storage of the indices to roughly $\log(D_B)$ bits for each bucket. For the original P^2 the index can now be stored in 2 bits per bucket.

6 Evaluation

We have implemented all these variants of histograms and compared their actual performance over a range of distributions. We have summarized the results in Figures 17 and 16.

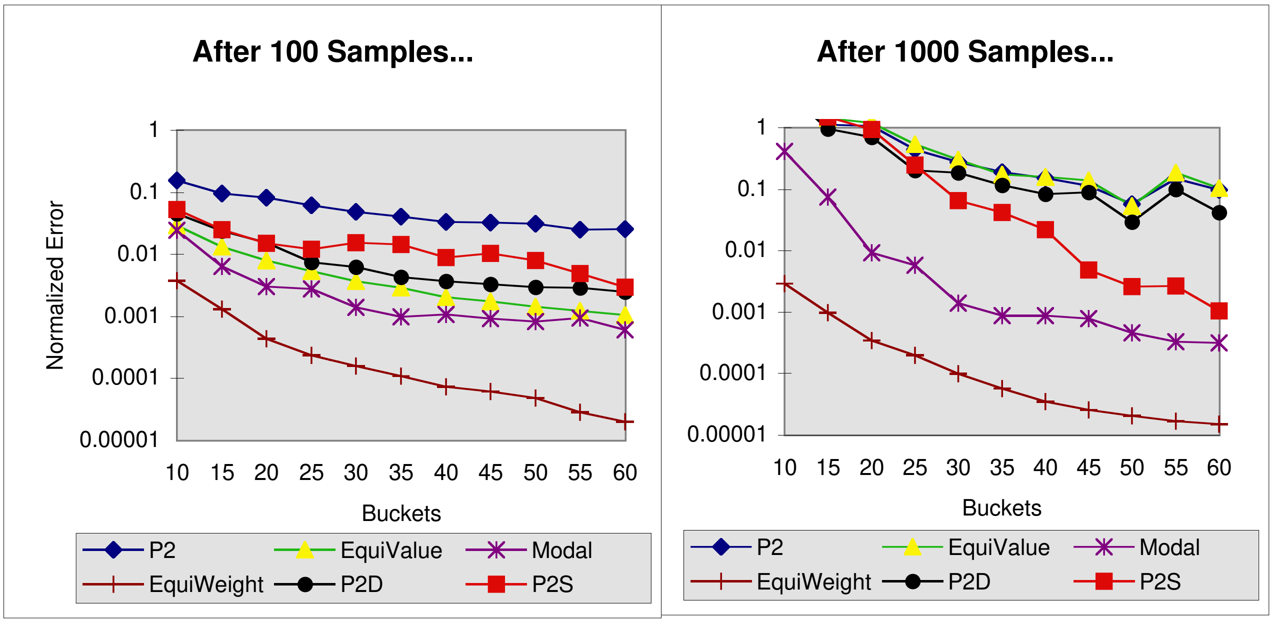


Figure 16: Accuracy of each histogram as a function of number of buckets

The implementations were designed to take `doubles` as data points. This is computationally more expensive and takes up more space than specializing for other data types (e.g. integers), but it allowed us to use a single implementation of each histogram for all the tests. We don't believe it significantly affects the *relative* timings, but caution against depending too much on the absolute performance numbers presented here. The times were measured on a 120 Mhz Pentium, running Linux 1.3.57. (Rankings using times measured on a Decstation 5000/240, running NetBSD 1.1, were similar).

For accuracy, we measure faithfulness to the sample statistics (the cumulative distribution defined by completely sorting *every* observation), rather than relative to the ideal underlying mathematical CDF (which *is* known in most of these user-designed test cases).

The computation of the error depends on the definition of `getQuantile`, above.

$$\text{Error} = \frac{1}{N}\sum_{i=0}^{N-1}\left|\text{Observations}_i - \text{getQuantile}\left(\frac{i}{N-1}\right)\right|$$

$$\text{Normalized Error} = \frac{\text{Error}}{\frac{1}{N}\sum_{i=0}^{N-1}\left|\text{Observations}_i - \overline{\text{Observations}}\right|}$$

where N is the total number of observations seen, `Observations` is a sorted array of every observation seen, and $\overline{\texttt{Observations}}$ is the sample mean.

We normalize `Error` to allow us to compare the accuracy of the histograms in recording different distributions[15].

To measure the accuracy of our histograms we compiled a test suite with a wide range of distributions. Some were continuous distributions (e.g. Normal, Exponential, ...) and their discrete counterparts (by taking `floor` of the continuous random variable). Others were randomly selected from a set of unrelated discrete values intended to model using the histogram to record the value of a variable in a program. Finally, a set of distributions were driven by actually timing functions with a microsecond clock. Each sample from a given distribution contained 10,000 observations. Each observation was stored into all of the histograms. We took 50 samples from each distribution in the test suite.

The normalized error reported in Figure 16 is the average of the MSE for each distribution, after the first 1000 observations were recorded.

Figure 17 shows the average cost (in microseconds) of inserting new data points into the various histograms described in this paper. We plot the speed against the accuracy of each histogram for a selected number of buckets.

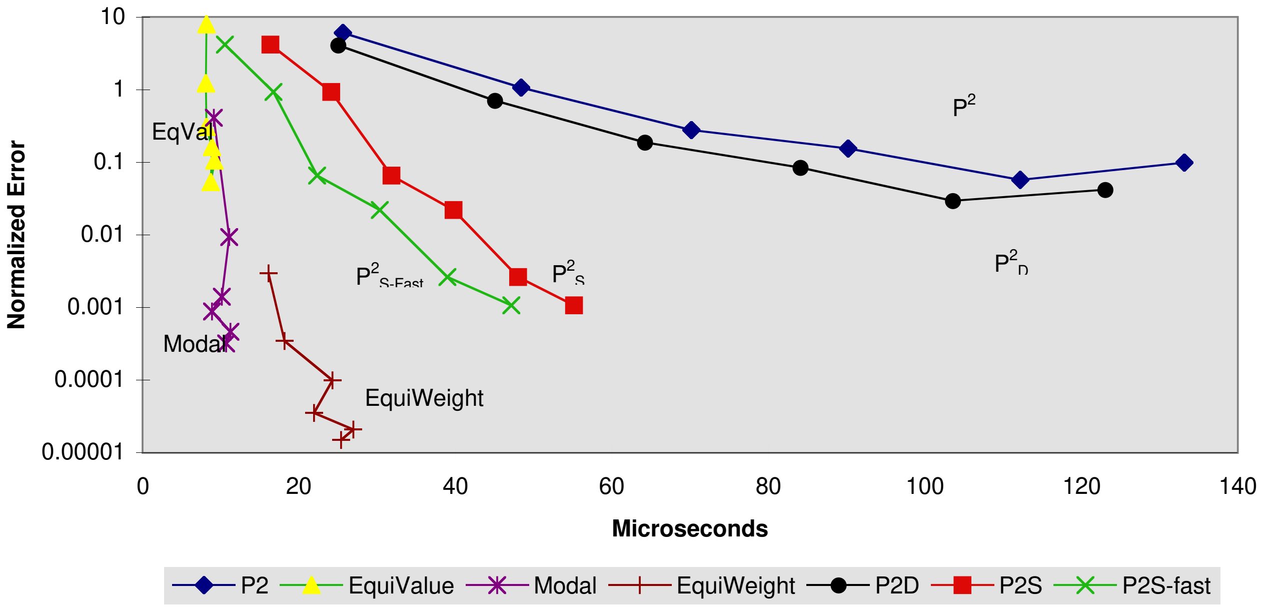


Figure 17: Graphical representation of average error after 1000 observations and performance cost of each histogram algorithms for 10 through 60 buckets, in 10 bucket increments.

The Appendix (Figures 18 (page 21) through 29 (page 23)) contains graphs that show the normalized error for each type of histogram for some representative distributions in the test suite. The normalized error is shown as a function of number of observations, inserted into a histogram with 20 buckets. The graphs show the MSE over all 50 runs for each test. Several interesting features about the accuracy of the histograms are shown here.

First, the available error is a good predictor of the *relative* error of each algorithm. Designing an algorithm that reduces the available error usually appears to reduce the measured error when actually run.

The equi-sized histograms appear to be an exception to this rule. They had much lower error than expected.

The graphs in the Appendix, though, appear to contradict this. In many of the graphs the error for equi-sized histograms is less than the error for the P^2 histograms. Yet, we claimed earlier that the available error was the same for equi-sized and equi-probable histograms. The resolution of this apparent paradox is that P^2

[15] We divide the error by N to allow us to compare histograms with different size samples. We divide by the area to compare against distributions with widely varying ranges, and we subtract out the average to keep the normalized error invariant over distributions that are just shifted by a constant amount.

histograms are not purely equi-probable, and thus their available error is worse than R/B. First, the points are interpolated and are not necessarily on the CDF. In particular, the interpolation is known to be worst in sparse buckets — which is precisely where the available error is largest. Second, in discrete distributions where a single value might appear enough times to fill more than one bucket, P^2 smooths out the CDF as it did in the Bernoulli example. Equi-sized histograms only recompute the bucket boundaries when the histogram bounds change (a new max or min is seen). For discrete distributions, or buckets with no hits at all, the equi-size histograms record this faithfully.

As expected, one can see that the discrepancy is most pronounced in distributions with long tails.

Second, distributions for "real" measurements (i.e. results of timing something with the clock, in the face of noise), seem ideally suited to equi-error and multi-modal histograms. The distributions are discrete with major gaps and multiple modes. The performance of the equi-error and multi-modal histograms can be hundreds of times more accurate than the other algorithms. it should be noted that most of the error occurs in the tails of the distribution where a small percentage of the data lies, but which has a significant effect on the average value.

Third, where modal histograms had to deal with erratically spaced discrete distributions, it performed worse than expected. Consider Figure 27 where the first 10 primes were chosen uniformly. They are spaced out over 28 values, so, although, equi-error, P^2_D, and P^2_S were able to identically recover the distribution, modal histograms interpolated non-prime integer values to fill out the curve, and so had error within a factor of 2 of equi-sized histograms. The behavior of P^2 in this example is indicative of a situation where there are fewer distinct values than buckets (see the error in the Bernoulli process). The error accumulates as P^2 adjusts the quantiles to non-existant values.

Finally it is interesting to notice how, in continuous distributions, P^2_S recovers from the gamble that it is worthwhile recording individual values for a while and then reverts to behavior identical to P^2 (Figures 18 and 20). Similarly, P^2_D abruptly shifts from discrete to continuous when the number of distinct values exceeds the number of buckets (Figure 23 shows this most clearly.)

6.1 Alternatives

Are self-scaling histograms the only solution?

We could consider using non-scaling histograms (fixed bucket size and lower bound[16]). This, however, can perform poorly. It would lead to either excessive memory overhead (too many buckets with a wide variation in bucket contents, some with few or no elements) or loss of information (too few buckets) — or even useless results, if you choose the size and low bound such that datum of interest are out of range!

An alternative solution is acquire the data twice, the first time just to get enough information to use "normal" histograms. However, you cannot always do this; consider meters in the kernel of an operating system or inside a big context dependent operation inside a long-running program or server. Sometimes it is no more practical to store all that information than to use self-scaling histograms in the first place. Consider a tool like `gprof` where you might be profiling hundreds of functions; if you wished to run the program twice to determine reasonable histogram bounds for each function, you would need to keep a reasonable number of observations. (If all you keep are the minimum, maximum and average, then a bimodal distribution will lose most of the info. Or, a couple of outliers might destroy your resolution near the center.) Even if you could successfully record enough information to determine reasonable histogram parameters, variations might change the bounds from run to run (consider a function which may or may not take a page fault)[17].

We conclude that self-scaling histograms are more widely applicable than competing approaches, assuming the computational overhead is low enough. The algorithms presented in this paper are good candidates for such low-overhead techniques.

6.2 Experience

Although these algorithms are being published here for the first time, earlier versions of the modal histograms have been used by the author over the course of 15 years to meter kernel performance in SWIFT [4], and the RVD (Remote Virtual Disks) network protocol, and were deployed in a general purpose metering[11, 12] bundled with the Symbolics Genera operating system for Lisp Machines. Over the course of that time, libraries employing modal histograms were written in C, Lisp, CLU, and C++. During that time these histograms

[16] A low bound of 0 for timing, and conservative, domain specific choices for other applications.

[17] We ignore the question of whether results that might not be repeatable are interesting.

have proved themselves invaluable in many instances of quick-and-dirty metering when time was not available to design more careful experiments. They have also served as one additional useful tool in all forms of system measurement and tuning.

7 Related Work

The basic work on self-scaling equi-probable histograms was done by Jain, et. al. [9] as an extension of the P^2 algorithm for computing quantiles. We have proposed several new algorithms for computing histograms, in particular multi-modal and equi-error histograms, each of which have greater accuracy and superior performance than P^2 for equivalent storage requirements. Our work stresses reconstructing the entire distribution. The P^2 algorithm is best suited for finding specific quantiles. In the cases where a set of quantiles *is* of interest, we have improved the accuracy and performance of P^2.

Interestingly, the field of Machine Learning tries to solve problems similar to ours. A large database of values is presented as input, and the machine learning systems try to develop a model that describes the data in a small number of terms.

Some interesting examples worth looking into are the EM Algorithm applied to classification (AutoClass), and the K-Means algorithm. See [5], [2], and [1]. These three attempt to classify or categorize a large body of data in terms of a smaller number of parameters. New data points are presented incrementally, and due to the large volume of data it is not feasible to store them all. Typically, they assume the data is a mixture of different distributions ("classes"). They continually refine their parameters until they arrive at the most likely distribution that would explain the observations they've seen. Autoclass, for example, searches the space of mixture models optimizing the number of classes, the type or description of each class, and the probability or weight that each class appears. The main points that distinguish their work from the self-scaling histograms described in this paper are, first, that they assume that the data can be parameterized as some sort of mixture of parameterized distributions, while we have seen that making that assumption loses some information that might be critical in understanding the behavior of your system. Further, they are less concerned about minimizing overhead (for example if we are recording the elapsed time of a function that is called thousands or millions of times a second). Their goal is to predictively fill in gaps in the data; ours is to compress the data without loss of critical information. Both attempt to characterize the data with absolutely no prior knowledge.

The algorithms we use to interpolate values between recorded data points are related to spline algorithms in computer graphics. Our problem is similar to choosing the optimal control points for a given monotonically increasing curve, where the number of control points is fixed. A fundamental difference is that we do not know the "real" curve: all we have are the previous set of control points and a random process telling us the "density" of the curve between two control points.

Similarly, the goodness of Equi-Error Histograms are related to a criteria known as *Importance Sampling* (the density of observations in a region should be proportional to the value of the function) in Monte Carlo Integration. The PDF is the derivative of the CDF, so finding the CDF is equivalent to integrating the PDF. Importance sampling would require the width of the bucket (dx) to be proportional to $1/\Delta$CDF, which would make the available error of each cell roughly equal. When this criteria is met, variance of the result is minimized. Equi-error histograms also try to minimize variance, but we are simultaneously trying to derive the shape of the function. In Monte Carlo Integration, the derivative of the function is known. Here, we are using the buckets not only to interpolate the function, but to discover its values. We have seen that for finite number of buckets equi-error buckets are not minimal.

8 Conclusions

We have presented accurate, efficient, and easy to use implementations of self scaling histograms. Equi-Error Histograms reproduce the distribution with very high accuracy, but have computational overhead proportional to the number of buckets. Multi-Modal histograms have very low overhead that is independent of the number of buckets, yet have accuracy that is comparable in many cases to Equi-Error histograms. Even when Multi-Modal Histograms have much larger error than Equi-Error histograms, the error is still much smaller than other competing techniques. It is possible to choose the best histogram to record data with no knowledge of the distribution of data you are measuring. In almost all cases the combination of lower computational cost and space overhead, combined with the relatively low error, argue for using Multi-Modal histograms.

When exceptional accuracy is required, and computational overhead isn't critical then Equi-Error histograms will acquire the data distribution more accurately for a given number of buckets. These considerations are independent of the values you are measuring, simply functions of external constraints.

Both equi-error and multi-modal histograms are significant improvements over existing techniques.

By recasting the problem of computing histograms in terms of minimizing the error when recovering the entire distribution, we were able to determine that accepted wisdom about the relative accuracy of equi-width and equi-probable histograms was incorrect. Further, this analysis led directly to a more efficient algorithm.

Finally, these algorithms should be useful in the common case of "large scale" metering. The algorithms presented in this paper should enable readers to consider using histograms next time (every time) they are about to instrument their code with a simple `count`, `average`, `stdev`, or `median`.

A Code

Many details of the algorithms in this paper were glossed over in the interest of space and clarity. C++ code implementing each of these histograms is available through the Web site at `http://www.dsg.stanford.edu`

B Figures

B.1 Tests comprising the average accuracy in Figures 17 and 16

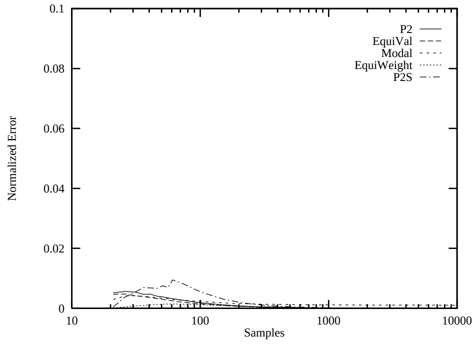


Figure 18: Mean Square Error compared to complete record of observations. Distribution = Uniform(0.0, 5.0)

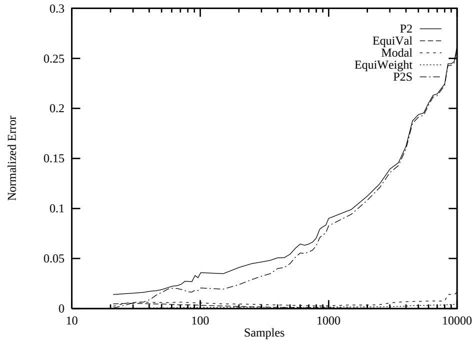


Figure 19: Mean Square Error compared to complete record of observations. Distribution = Exponential, mean = 5.0

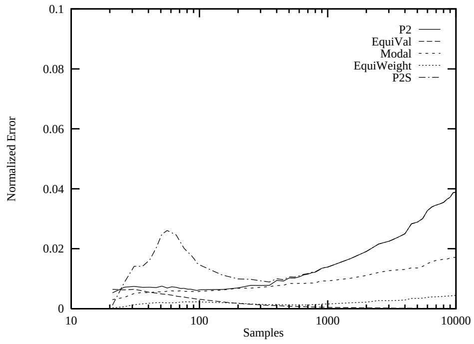


Figure 20: Mean Square Error compared to complete record of observations. Distribution = Normal(5.0, 2.0)

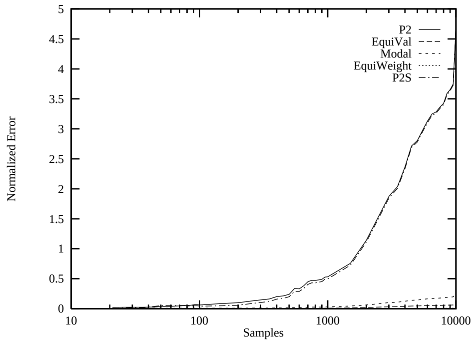


Figure 21: Mean Square Error compared to complete record of observations. Distribution = Lognormal, mean=5.0, stdev = 1.0

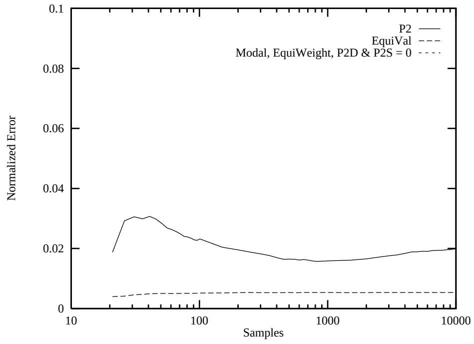


Figure 22: Mean Square Error compared to complete record of observations. Distribution = Discrete Uniform(0.0, 5.0)

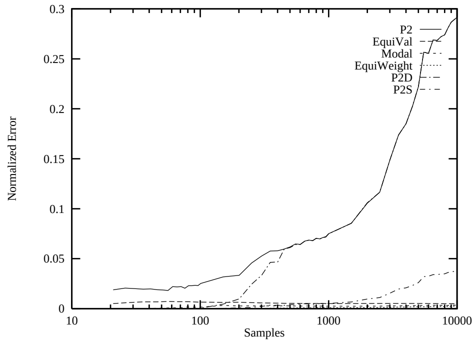


Figure 23: Mean Square Error compared to complete record of observations. Distribution = Discrete Exponential, mean = 5.0

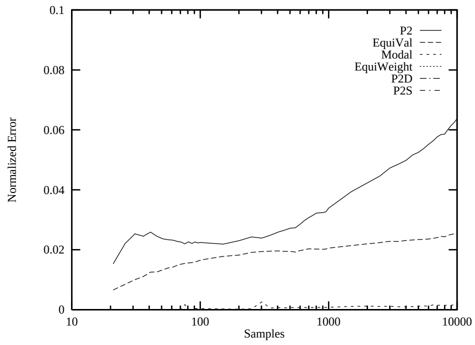


Figure 24: Mean Square Error compared to complete record of observations. Distribution = Discrete Normal(5.0, 2.0)

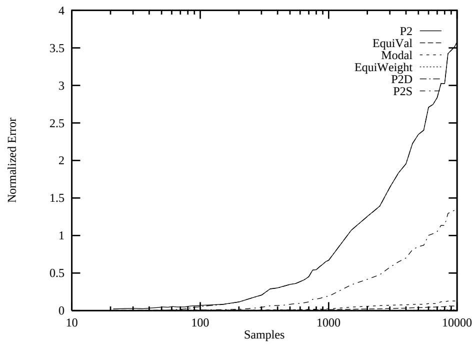


Figure 25: Mean Square Error compared to complete record of observations. Distribution = Discrete Lognormal, mean=5.0, stdev = 1.0

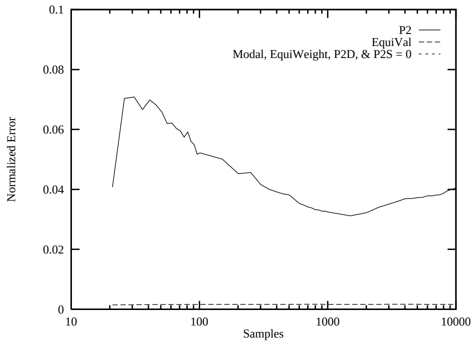


Figure 26: Mean Square Error compared to complete record of observations. Distribution = Bernoulli Process

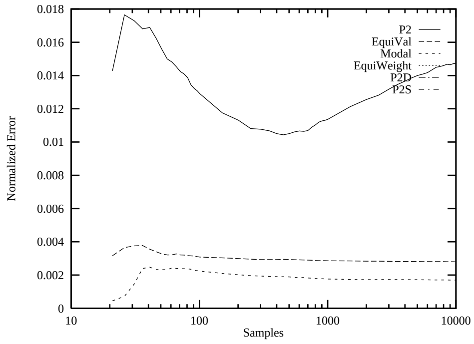


Figure 27: Mean Square Error compared to complete record of observations of uniformly choosing among the first 10 prime numbers.

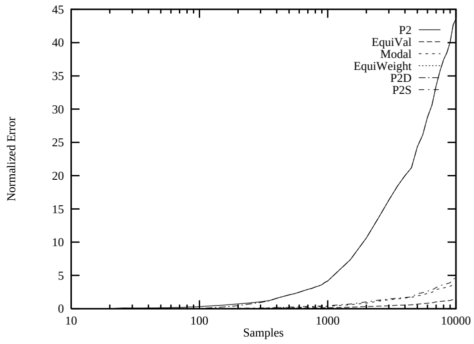


Figure 28: Mean Square Error compared to complete record of observations of measuring the clock overhead on Decstation 5000/240 running NetBSD 1.1.

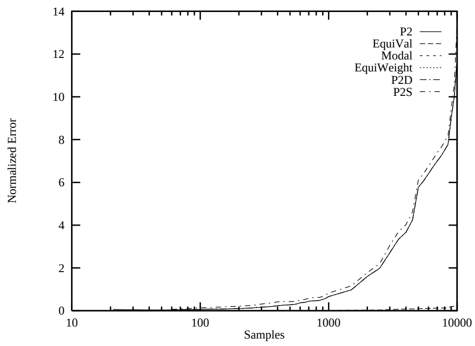


Figure 29: Mean Square Error compared to complete record of observations. of measuring the cost of insertion into the P^2 histogram on Decstation 5000/240 running NetBSD 1.1.

References

[1] J.R. Anderson and M. Matessa, Explorations of an Incremental, Bayesian Algorithm for Categorization, *Machine Learning*, 9, 275-308, Boston 1992

[2] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, AutoClass: A Bayesian Classification System, *Proceedings of the Fifth International Conference on Machine Learning*, (pp. 54-64) San Mateo, CA, June 12-14, 1988.

[3] D.R. Cheriton and K. Duda. The Cache Kernel: a fault-tolerant hardware adaptation layer. OSDI '94.

[4] D.D. Clark. "The Structuring of Systems Using Upcalls", *10th ACM SOSP*, Orcas Island, WA, 1985

[5] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Society, Series B*, 39(1):1-38, 1977.

[6] D. Ferrari. *Measurement and Tuning of Computer Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[7] D. Gladstein, Technical Correspondence, *Communications of the ACM*, Vol. 29, no 6. 557-558, Jun 86.

[8] S.L. Graham, P.B. Kessler, M.K. McKusick, "gprof: A Call Graph Execution Profiler", *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp 120-126, June 1982.

[9] R. Jain and I. Chlamtac. The P^2 Algorithm for Dynamic Calculation of Quantile and Histograms Without Storing Observations., *Communications of the ACM*, Vol. 28, no 10. 1076-1085, Oct 85.

[10] R. Jain. *Art of Computer Systems Performance Analysis*, Wiley 1991.

[11] *Symbolics Lisp Machine Manual*, Genera 7.2 Release notes, Symbolics Inc. Cambridge MA, Feb 1988.

[12] *Symbolics Lisp Machine Manual*, Book 12, Program Development Utilities, Symbolics Inc. Burlington MA, Feb. 1990.

[13] D. Walter, Technical Correspondence, *Communications of the ACM*, Vol. 29, no 12. 1241-1242, Dec 86.