# COMPUTING EXTREMELY ACCURATE QUANTILES USING $t$-DIGESTS

TED DUNNING AND OTMAR ERTL

Abstract. Two variants of an on-line algorithm for computing approximations of rank-based statistics are presented that allow controllable accuracy, particularly near the tails of the distribution. Moreover, this new algorithm can be used to compute hybrid statistics such as trimmed means in addition to computing arbitrary quantiles. An unusual property of the method is that it allows a quantile $q$ to be computed with an accuracy relative to $\max(q, 1-q)$ rather than with an absolute accuracy as with most methods. This new algorithm is robust with respect to highly skewed distributions or highly ordered datasets and allows separately computed summaries to be combined with no loss in accuracy.

An open-source Java implementation of this algorithm is available from the author. Implementations in Go and Python are also available.

## 1. Introduction

Given a set of numbers, it is often desirable to compute rank-based statistics such as the median, 95-th percentile or trimmed means in an on-line fashion. In many cases, there is an additional requirement that only a small data structure needs to be kept in memory as data is processed in a streaming fashion. Traditionally, such statistics have been computed by sorting the data and then either finding the quantile of interest by interpolation or by re-processing all samples within particular quantile ranges. This sorting approach can be infeasible for very large datasets or when quantiles of many subsets must be calculated. This infeasibility has led to interest in on-line approximate algorithms. Previous algorithms can compute approximate values of quantiles using constant or only weakly increasing memory footprint, but these previous algorithms cannot provide constant relative accuracy. The new algorithm described here, the $t$-digest, provides constant memory bounds and constant relative accuracy while operating in a strictly on-line fashion.

### 1.1. Previous work.

One early algorithm for computing on-line quantiles is described by Chen, Lambert and Pinheiro in [CLP00]. In that work specific quantiles were computed

by incrementing or decrementing an estimate by a value proportional to the simultaneously estimated probability density at the desired quantile. This method is plagued by a circularity in that estimating density is only possible by estimating yet more quantiles. Moreover, this work did not allow the computation of hybrid quantities such as trimmed means.

Munro and Paterson[MP80] provided an alternative algorithm to get a precise estimate of the median. This is done by keeping $s$ samples from the $N$ samples seen so far where $s << N$ by the time the entire data set has been seen. If the data are presented in random order and if $s = \theta(N^{1/2} \log N)$, then Munro and Paterson's algorithm has a high probability of being able to retain a set of samples that contains the median. This algorithm can be adapted to find a number of pre-specified quantiles at the same time at proportional cost in memory. The memory consumption of Munro-Paterson algorithm is, however, excessive if precise results are desired. Approximate results can be had with less memory, however.

A more subtle problem is that the implementation of Munro and Paterson's algorithm in Sawzall[PDGQ05] and the Datafu library[Lin] uses a number of buckets computed from the GCD of the desired quantiles. This means that if you want to compute the 99-th, 99.9-th and 99.99-th percentiles, a thousand buckets are required, each of which requires the retention of many samples. We will refer to this implementation of Munro and Paterson's algorithm as MP01 in results presented here.

One of the most important results of the work by Munro and Paterson was a proof that computing any particular quantile exactly in $p$ passes through the data requires $\Omega(N^{1/p})$ memory. For the on-line case, $p = 1$, which implies that on-line algorithms cannot guarantee to produce the precise value of any particular quantile. This result together with the importance of the on-line case drove subsequent work to focus on algorithms to produce approximate values of quantiles.

Greenwald and Khanna[GK01] provided just such an approximation algorithm that is able to provide estimates of quantiles with controllable accuracy. This algorithm (which we shall refer to as GK01 subsequently in this paper) requires less memory than Munro and Paterson's algorithm and provides approximate values for pre-specified quantiles.

An alternative approach is described by Shrivastava and others in [SBAS04]. In this work, incoming values are assumed to be integers of fixed size. Such integers can trivially be arranged in a perfectly balanced binary tree where the leaves correspond to the integers and the interior nodes correspond to bit-wise prefixes. This tree forms the basis of the data

structure known as a Q-digest. The idea behind a Q-digest is that in the uncompressed case, counts for various values are assigned to leaves of the tree. To compress this tree, sub-trees are collapsed and counts from the leaves are aggregated into a single node representing the sub-tree such that the maximum count for any collapsed sub-tree is less than a threshold that is a small fraction of the total number of integers seen so far. Any quantile can be computed by traversing the tree in left prefix order, adding up counts until the desired fraction of the total is reached. At that point, the count for the last sub-tree traversed can be used to interpolate to the desired quantile within a small and controllable error. The error is bounded because the count for each collapsed sub-tree is bounded.

The salient virtues of the Q-digest are

- the space required is bounded proportional to a compression factor $k$
- the maximum error of any quantile estimate is proportional to $1/k$ and
- the desired quantiles do not have to be specified in advance.

On the other hand, two problems with the Q-digest are that it depends on the set of possible values being known in advance and produces quantile estimates with constant error in $q$. In practice, this limits application of the Q-digest to samples which can be identified with the integers. Adapting the Q-digest to use a balanced tree over arbitrary elements of an ordered set is difficult. This difficulty arises because rebalancing the tree involves sub-tree rotations and these rotations may require reapportionment of previously collapsed counts in complex ways. This reapportionment could have substantial effects on the accuracy of the algorithm and in any case make the implementation much more complex because the concerns of counting cannot be separated from the concerns of maintaining a balanced tree.

1.2. **New contributions.** The work described here introduces a new data structure known as the $t$-digest which is the result of a clustering of real-valued samples. The $t$-digest differs from more well-known forms of clustering such as $k$-means in that the samples are from $\mathbb{R}^1$ rather than taken from any space with a metric and because the clusters are limited by size in a special way. The $t$-digest differs from previous structures designed for computing approximate quantiles in several important respects. First, although data is clustered and summarized in the $t$-digest, the range of data included in different clusters may overlap. Second, the bins are summarized by a centroid value and an accumulated weight representing the number of samples contributing to a bin. Third, the samples are accumulated in such a way that only a few samples contribute to bins corresponding to

extreme quantiles so that relative error is bounded instead of maintaining constant absolute error as with previous methods.

With the $t$-digest, accuracy for estimating the $q$ quantile is constant relative to $q(1-q)$. This is in contrast to earlier algorithms which had errors independent of $q$. The relative error bound of the $t$-digest is convenient when computing quantiles for $q$ near 0 or 1 as is commonly required. As with the Q-digest algorithm, the accuracy/size trade-off for the $t$-digest can be controlled by setting a single compression parameter $\delta$ with the amount of memory required proportional only to $\Theta(\delta)$.

## 2. The $t$-digest

A $t$-digest is generated by clustering real-valued samples and retaining the mean and number of samples for each cluster. This clustering can then be used to estimate quantile-related statistics with particularly high accuracy near the tails of a distribution. Algorithmically, there are two important ways to form a $t$-digest from a set of numbers. One version keeps a buffer of incoming samples. Periodically, this buffer is sorted and merged with the centroids computed from previous samples. This merging form of the $t$-digest algorithm has the virtue of allowing all memory structures to be allocated statically. On an amortized basis, this buffer-and-merge algorithm can be very fast especially if the input buffer is large. The other major $t$-digest algorithm is more akin to traditional clustering algorithms where new samples are added one at a time to whichever cluster is nearest. Both algorithms are described here and implementations for both are widely available.

### 2.1. **The basic concept.**

Take a $n$ samples $\{x_1 \dots x_n\} = X \subset \mathbb{R}^1$. Define a digest as a partition of $X$ consisting of sets of samples $\pi_i \subset X$. The subsets $\pi_i$ in this partition are referred to as clusters. The number of samples in a cluster $\mathcal{C}$ is written as $|\mathcal{C}|$ and is typically referred to as the weight of the cluster.

Each cluster $\mathcal{C}$ in a digest has $|C| > 0$ samples and an associated mean $\overline{\mathcal{C}} = \sum_{x \in \mathcal{C}} x / |\mathcal{C}|$. The clusters in the digest can be partially ordered according to their means. Clusters with identical means are assigned an arbitrary fixed ordering so that we can index the clusters consistently. Given this complete ordering, we define a left and right weight for each cluster

$\mathcal{C}_i$ as the sum of the weights of clusters to the left and right of $\mathcal{C}_i$. That is

$$\mathcal{W}_{\text{left}}(\mathcal{C}_i) = \sum_{j<i} |C_j|$$
$$\mathcal{W}_{\text{right}}(\mathcal{C}_i) = \sum_{j>i} |C_j|$$

Note that so far, we have no constraint about the ordering of the elements in different point sets. We refer to a digest as strongly ordered if $i > j \implies x \geq y$ for $x \in \pi_i$ and $y \in \pi_j$. We refer to a digest as weakly ordered if $i > j \implies x > \overline{\mathcal{C}}_j$ for $x \in \pi_i$.

Such a digest is a $t$-digest if every cluster has unit weight, or has weight bounded using a scale function as defined below. A $t$-digest is called fully merged if no two consecutive clusters can be combined without violating the weight bound.

2.2. **A simple start.** Suppose that all of the samples $X = x_1 \dots x_n$ are presented in ascending order with ties broken arbitrarily. Since the samples are ordered, we can use the index of each sample to determine the value of the empirical quantile for any new value.

If we form a trivial digest where each cluster has a single point, this digest will be a strongly ordered $t$-digest, but will likely not be fully merged.

We can group consecutive samples greedily from left to right into sub-sequences of samples, $X = \{s_1|s_2|\dots|s_m\}$ where $s_i = \{x_{\texttt{left}(i)} \dots x_{\texttt{right}(i)}\}$ such that each $s_i$ has as many samples as possible subject to the size bound. This will also be a strongly ordered $t$-digest, and will also be fully merged.

The key idea of the $t$-digest is that the size of each sub-sequence is chosen so that the sub-sequence is small enough to get accurate quantile estimates by interpolation, but large enough so that we don't wind up with too many sub-sequences. Importantly, we force sub-sequences near both ends to be small while allowing sub-sequences in middle to be larger in order to get fairly constant relative accuracy.

2.3. **The size bound for clusters.** The size bound for $t$-digests is based on the idea of a scale function that forces clusters near the beginning or end of the digest to be small, possibly containing only a single sample. The scale function is chosen to provide an appropriate trade-off between very accurate quantile estimate in the tails of a distribution, reasonable accuracy near the median while keeping the number of clusters as small as possible.

To limit the sub-sequence size in this way, we define a non-decreasing function from quantile $q$ to a notional index $k$ with compression parameter $\delta$. This mapping is known as the scale function. A one common scale function for the $t$-digest is

$$k_1(q) = \frac{\delta}{2\pi}\sin^{-1}(2q-1)$$

As with any scale function, $k_1$ is non-decreasing. It has minimum value $k(0) = -\delta/4$ and maximum value $k(1) = \delta/4$. Figure 1 shows the relationship between $k$ and $q$ for $\delta = 10$. In this figure, the horizontal lines are spaced uniformly at integer values of $k$. Vertical lines

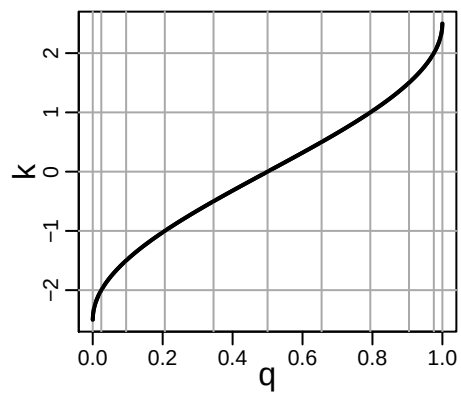


FIGURE 1. The scale function translates the quantile $q$ to the scale factor $k$ in order to give variable size steps in $q$. Limiting cluster sizes allows better accuracy near $q = 0$ or $q = 1$.

are drawn from the intersection of the scale function with these evenly spaced horizontals. Note how much closer together these vertical lines are near $q = 0$ and $q = 1$.

The scale function provides the necessary mechanism to define the size bound of a $t$-digest. Every cluster $\mathcal{C}$ with more than one sample should have the $k$-size (written as $|\mathcal{C}|_k$) at most 1,

$$\begin{aligned} q_{\text{left}} &= \mathcal{W}_{\text{left}}(\mathcal{C})/n \\ q_{\text{right}} &= q_{\text{left}} + |\mathcal{C}|/n \\ |\mathcal{C}|_k &= k\left(q_{\text{right}}\right) - k\left(q_{\text{left}}\right) \leq 1 \end{aligned}$$

In a fully merged $t$-digest adjacent clusters, cannot be merged because the result would be too big

$$|\mathcal{C}_i \cup \mathcal{C}_{i+1}|_k = |\mathcal{C}_i|_k + |\mathcal{C}_{i+1}|_k > 1$$

Together, these conditions imply that for a fully merged $t$-digest using $k_1$ with at least $\delta$ samples, the number of clusters $m$ is in the range $\lfloor \delta/2 \rfloor \leq m < \lceil \delta \rceil$.

2.4. **The effect of a scale function.** The only really necessary characteristic of a scale function up to now is that be non-decreasing. As such, a linear scale function $k_0(q) = \delta q$ can be used instead of $k_1$. Using a linear scale function results in nearly uniform cluster sizes and constant absolute error in $q$ which is similar to the behavior of most previously reported quantile approximation algorithms.

Figure 2 shows how estimating $q$ near tails is improved for a strongly ordered, fully merged $t$-digest constructed with a scale function that keeps clusters small near extreme values of $q$. This figure shows roughly the first percentile of $10,000$ data points sampled

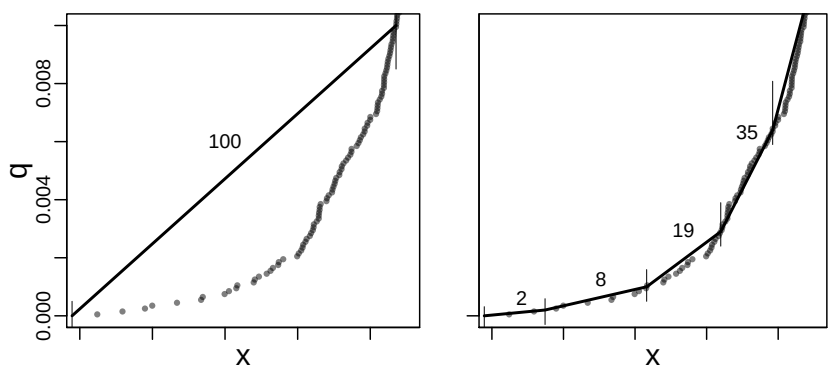


FIGURE 2. The left panel shows linear interpolation of the cumulative distribution function near $q = 0$ with 100 equal sized bins applied to $10,000$ data points sampled from an exponential distribution. The right panel shows the same interpolation with variable size bins as given by a $t$-digest with $\delta = 100$. The numbers above the data points represent the number of points in each bin.

using $x \sim \log u$ where $u \sim \text{Uniform}(0, 1)$. In the left panel, the data points have been divided into 100 bins, each with 100 data points, of which only the left-most bin is visible. The use of equal sized bins means that the interpolated value of $q$ in the left panel of the figure has a substantial and unavoidable error. The right panel, on the other hand, shows a $t$-digest with roughly the same number of bins (102 instead of 100), but with many fewer points in bins near the $q = 0$. A value of $\delta = 100$ is used here which is quite typical in practical use.

Of course, since every sample must be in some bin, filling some bins with fewer than 100 samples requires that other bins must have more than 100 samples or we have to have more bins. Note, however, that having fewer samples in the bins near $q = 0$ or $q = 1$ improves accuracy by orders of magnitude, while increasing the bin sizes near $q = 1/2$ degrades accuracy to a much smaller degree. In this particular example, the first bin has only 2 samples and thus zero error and the second bin has only 10 samples giving roughly 100 times smaller error than a cluster of 100 samples would give. The clusters near $q = 1/2$, on the other hand, have about 1.6 times more samples than the uniform case, increasing errors by a factor of just over two relative to equal sized bins. The overall effect is that quantile estimation accuracy is dramatically improved at the extremes but only modestly impaired near the median.

2.5. **Merging independent $t$-digests.** In the previous section, the algorithm for forming a $t$-digest took a set of samples as inputs. Nothing, however, would prevent the algorithm from being applied to a set of weighted samples. As long as individual weights are smaller than the size limit imposed by the scale function, the result can still be a well-formed $t$-digest.

If we form independent $t$-digests $t_X$ and $t_Y$ from separate sequences $X$ and $Y$, these $t$-digests can clearly be used to estimate quantiles of $X \cup Y$ by separately computing quantiles for $X$ and $Y$ and combining the results. It is much more useful, however, if we choose the scale factor so that it guarantees that the ordered union of two $t$-digests will still be a valid, though probably not fully merged, $t$-digest. The scale factors $k_0$ and $k_1$ both have this property as do the other scale functions described later.

The digest formed by merging two digests will not, however, typically be fully merged. We can make it so by merging the centroids wherever consecutive clusters can be combined and still meet the size bound. The resulting $t$-digest will not generally be the same as if we had computed a $t$-digest $t_{X \cup Y}$ from all of the original data at once even though it will meet the same size constraint. In particular, even if $t_X$ and $t_Y$ are strongly ordered, their union $t_X \cup t_Y$ may not even be weakly ordered. This can happen, for instance, when there are centroids in very nearly the same position in the two digests being merged.

The observation that $t$-digests formed by merging other digests will produce accurate quantile estimates has substantial empirical support, even with highly structured data sets such as ordered data or data with large numbers of repeated values, but there is, as yet, no rigorous proof of any accuracy guarantees. The size bounds are, nevertheless, are proved.

The ability to merge $t$-digests makes parallel processing of large data-sets relatively simple since independent $t$-digests can be formed from disjoint partitions of the input data and then combined to get a $t$-digest representing the complete data-set.

A similar divide and conquer strategy can be used to allow $t$-digests to be used in OLAP systems. The idea is that queries involving quantiles of subsets of data can be approximated quickly by breaking the input data set into subsets corresponding to each unique combination of selection factors. A single $t$-digest is then pre-computed for each unique combination of selection factors. To the extent that all interesting queries can be satisfied by disjoint unions of such primitive subsets, the corresponding $t$-digests can be combined to compute the desired result.

2.6. **Progressive merging algorithm.** The observation that merging $t$-digests gives good accuracy suggests a practical algorithm for constructing a $t$-digest from a large amount of data. The basic idea is to collect data in a buffer. When the buffer fills, or when a final result is required, sort the bufferful of new data together with any previously created centroids, and make a single pass, merging points or centroids together whenever the size limits can be satisfied by the merged value. With an arbitrarily large buffer, this algorithm reduces to the original approach for constructing a $t$-digest from sorted data since the overall effect is simply a single merging pass through all the data. For smaller buffer sizes, however, many merge passes are required to process a large amount of data. This is similar to forming independent $t$-digests on buffers of data and then sequentially merging them to get the final result.

The operation of merging a buffer's worth of samples into an existing set of centroids is shown in Algorithm 1. Note that the check on the size bound is optimized to only require only as many evaluations of $k(q)$ during the merge as there are output values in $C'$. This is done by computing the bounding value $q_{\text{limit}} = k^{-1}(k(q) + 1)$ each time a new centroid is emitted. This allows the conditional to be triggered based on comparisons of $q$ so that if many points are merged, no additional calls to $k$ are needed. If $n \gg 2\lceil\delta\rceil$, this can result in a considerable speedup since computing $\sin^{-1}$ is expensive. Note also that this algorithm allows static allocation of all data structures as four arrays of primitive values, avoiding all dynamic allocation and structure boxing/unboxing.

The run-time cost of the merge variant of the $t$-digest is a mixture of the frequent buffer inserts and the rare merges. The buffer inserts are very fast since they consist of an array write, index increment and an overflow test. The merges consist of the buffer

**Algorithm 1:** Merging new data into a $t$-digest

**Input**: Sequence $C = [c_1 \dots c_m]$ a $t$-digest containing real-valued, weighted centroids with components `sum` and `count` arranged in ascending order by mean, data buffer $X = x_1, \dots x_n$ of real-valued, weighted points, and compression factor $\delta$

**Output**: New ordered set $C'$ of weighted centroids forming a $t$-digest

```
X ← sort(C ∪ X);
S = Σ_i x_i.count;
C' = [], q_0 = 0;
q_limit = k^{-1}(k(q_0) + 1);
σ = x_1;
for i ∈ 2 ... (m + n) :
    q = q_0 + (σ.count + x_i.count)/S;
    if q ≤ q_limit :
        σ ← σ + x_i;
    else:
        C'.append(σ);
        q_0 ← q_0 + σ.count/S;
        q_limit ← k^{-1}(k(q_0, δ) + 1, δ);
        σ ← x_i;
C'.append(σ);
return C'
```

sort and the merge itself. The merge involves a scan through both the buffer and the existing centroids plus a number of calls to the scale function roughly equal to the size of the result which is bounded by $\lceil \delta \rceil$ for common scale functions. If $c_1$ is the input buffer size, the dominant costs are the sort and the scale function calls so the amortized cost per input value is roughly $C_1 \log c_1 + C_2 \lceil \delta \rceil / c_1$ where $C_1$ and $C_2$ are parameters representing the sort and scale function costs respectively. This overall amortized cost has a minimum where $c_1 \approx \delta\, C_2/C_1$. The constant of proportionality should be determined by experiment, but micro-benchmarks indicate that $C_2/C_1$ is in the range from 5 to 20 for a single core of an Intel i7 processor. In these micro-benchmarks, increasing the buffer size to $10\lceil \delta \rceil$ dramatically improves the average speed but further buffer size increases have much less effect.

Further optimization in the case of the scale function $k_1$ is possible by speeding up the evaluation of $\sin^{-1}$. It is difficult to build a high quality approximation of $\sin^{-1}$ over the entire domain $[0, 1]$ without using other high cost functions such as a square root, but by

limiting the domain where the approximation is used to $q \in [\epsilon, 1-\epsilon]$, where $\epsilon \approx 0.01$, simple and fast approximations are available.

2.7. **The clustering variant.** If we allow the buffer in the merging variant of the $t$-digest algorithm to contain just a single element so that merges take place every time a new point is added, the algorithm takes on a new character and becomes much more like clustering than buffering and merging.

The basic outline of the clustering algorithm for constructing a $t$-digest is quite simple. An initially empty ordered list of centroids, $C = [c_1 \dots c_m]$ is kept. Each centroid consists of a mean and a count. To add a new value $x_n$ with a weight $w_n$, the set of centroids is found that have minimum distance to $x_n$. This set is reduced by retaining only centroids whose $k$-size after adding $w_n$ would meet the size bound. If more than one centroid remains, the one with maximum weight is selected. If a acceptable centroid is found, then the new point, $(x_n, w_n)$, is added to that centroid. If no satisfactory centroid is found then $(x_n, w_n)$ is used to form a new centroid with weight $w_n$ and the next point is considered.

This clustering variant is shown more formally as Algorithm 2.

**Algorithm 2:** Construction of a $t$-Digest by clustering

**Input**: Ordered set of weighted centroids $C = \{\}$, sequence of real-valued, weighted points $X = \{(x_1, w_1), \dots (x_N, w_N)\}$, and accuracy tolerance $\delta$

**Output**: final set $C = [c_1 \dots c_m]$ of weighted centroids

```
for (x_n, w_n) ∈ X :
    z = min |c_i.mean − x|;
    S = {c_i : |c_i.mean − x| = z ∧ |c_i + w_1|_k < 1};
    if |S| > 0 :
        S.sort(key = λ(c){−c.sum});
        c ← S.first() ;
        c.count ← c.count + w_n;
        c.mean ← c.mean + (x_n − c.mean)/c.sum;
    else:
        C ← C + (x_n, w_n);
    if |C| > Kδ :
        C ← merge(C, {});
return C
```

In this algorithm, a centroid object contains a mean and and a count. Updating such an object with a new data-point $(x, w)$ is done using Welford's method [Wik, Knu98, Wel62].

As shown here, the number of points assigned to each cluster is limited so no cluster exceeds the $t$-digest size constraint, but certain insertion orders can cause the number of centroids to increase without bound. If the values of $X$ are in ascending or descending order, for instance, then $C$ will contain as many centroids as samples inserted. This will happen because each new value of $X$ will always form a new centroid because each new point will be the new minimum or maximum. To avoid this pathology, if the number of centroids becomes excessive, the entire set of centroids can be consolidated using the merge step in Algorithm 1.

2.8. **Alternative scale functions.** The $k_1$ and $k_0$ scale functions that we have mentioned are not the only ones possible. In fact, there are two additional important functions that provide important accuracy/size trade-offs. Altogether, these four functions serve as useful scale functions

$$
\begin{aligned}
k_0(q) &= \frac{\delta}{2} q \\
k_1(q) &= \frac{\delta}{2\pi} \sin^{-1}(2q-1) \\
k_2(q) &= \frac{\delta}{Z(n)} \log \frac{q}{1-q} \\
k_3(q) &= \frac{\delta}{Z(n)} \begin{cases} \log 2q & \text{if } q \le 1/2 \\ -\log 2(1-q) & \text{if } q > 1/2 \end{cases}
\end{aligned}
$$

2.9. **Interpolation of the cumulative distribution function.** The information in a fully-merged $t$-digest is not sufficient, in general, to precisely determine the empirical quantile $q$ of any particular value $x$ because information is lost as samples are clustered together. The only exception to this is in the case of clusters containing only a single sample where the centroid is the same as the single sample.

The approach taken in current implementations of the t-digest is to make several assumptions about the distribution of samples in the original data. These are:

(1) The samples for a cluster are evenly split into samples that are to the left of the centroid and those to the right if there is more than one sample.
(2) The samples between two clusters are uniformly distributed between them.

The first assumption is obviously the best we can do with no further information about the expected distribution. The second assumption is true asymptotically for continuous distributions as the number of clusters in a $t$-digest increases if the samples associated with each cluster do not extend beyond the neighboring centroids. In practice, clusters in a $t$-digest are sufficiently localized and quantiles even for highly skewed distributions can be computed with satisfactory accuracy.

Based on these assumptions, we have four important cases. These are a) interpolation between clusters that each have more than one sample, b) interpolation between a multi-sample cluster and a cluster with just a single sample, c) interpolation between clusters each with just a single sample and d) interpolation for the first and last cluster.

Between two clusters where both have more than one sample, the empirical cumulative distribution function is approximated using a linear approximation between two consecutive centroids that allocates half the weight of each centroid to the interval between them. This is illustrated in Figure 3. This method of interpolation simply sets the quantile for multi-

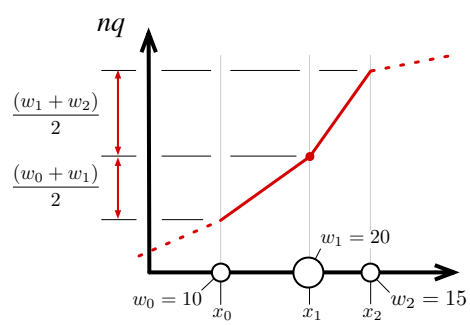


FIGURE 3. Interpolation of the empirical cumulative distribution function between centroids of clusters with more than one sample is done by assuming half of the points for each centroid are to the left of the centroid and half are to the right.

sample clusters so that the quantile is the mid-point of the cluster at the value of the centroid.

Such an interpolation is not as accurate as we would like, however, where clusters have a single sample so we adopt the methods shown in Figure 4. In such a case, we know that this single sample is located exactly at the centroid for that cluster. As such, we know that the true value of the distribution function at the centroid as well as infinitessimally to the left and right. When the neighboring cluster has multiple samples, this knowledge can be

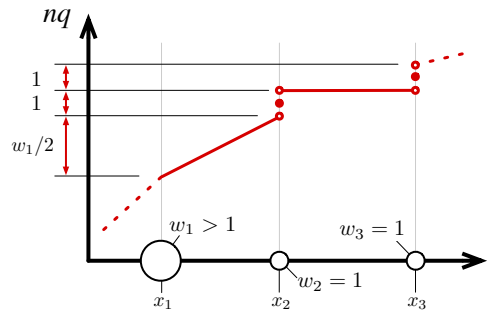


FIGURE 4. Clusters with a single sample are handled specially. Adjacent to a normal cluster, as is the case between $x_1$ and $x_2$, interpolation is done assuming $w_1/2$ samples occur between $x_1$ and $x_2$, and the single sample at $x_2$. That single sample causes the cumulative distribution to step to the mid-point of the individual sample at $x_2$. Between singleton clusters $x_2$ and $x_3$, the cumulative distribution is given a constant value until it steps again at $x_3$.

combined with interpolation as before, but when the next cluster is itself a singleton, we can avoid interpolation entirely.

The final case to be considered where a cluster is either the first or last cluster in the digest. When such a terminal cluster has only one sample, then we know that $x_{\max}$ is due exactly to the one sample in that cluster. Where the cluster has exactly two samples, then we can treat the final cluster as two singletons since we know that the two samples were at $x_{\max}$ and $x_n - (x_{\max} - x)$ and thus they can be treated as two singletons. Where there are more than two samples, however, it is advantageous to treat the cluster as having half the samples before the centroid and half after. For the half of the samples that come after, one of those samples must be at $x_{\max}$ and thus it can be treated as a singleton with the remainder being interpolated. All of this discussion is reversed, of course, first for last and left for right when treating the first cluster. This is shown in Figure 5 for the last cluster in a digest.

For scale functions $k_2$ and $k_3$ the first and last clusters will always be singletons and thus will never need this special handling. For the $k_1$ scale function, however, the first and last clusters may have more than one sample making it desirable to interpolate out beyond the centroid.

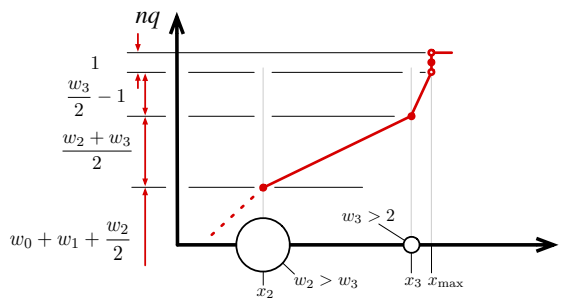


FIGURE 5. For the last clusters, if the cluster weight is greater than 2, we can use the fact that a singleton must occur at $x_{\min}$ or $x_{\max}$ to improve interpolation accuracy. The opposite is done for the first cluster.

2.10. **Accuracy.** The scale function $k_1$ defined earlier is not the only one that can be used. One particularly simple scale function is the linear scale $k_0$ which is defined as

$$k_0(q) = \frac{\delta}{2} q$$

This scale function provides uniform cluster size and thus does not emphasize accuracy near the tails. Digests constructed using $k_0$ provide an alternative to standard quantile estimation algorithms.

Other important scale functions allow even greater emphasis on the tails of the distribution than $k_1$. These include

$$k_2(q) = \frac{\delta}{Z(n)} \log \frac{q}{1-q}$$

$$k_3(q) = \frac{\delta}{Z(n)} \begin{cases} \log 2q & \text{if } q \leq 1/2 \\ -\log 2(1-q) & \text{if } q > 1/2 \end{cases}$$

Both $k_2$ and $k_3$ grow without bound at $q = 0$ or $q = 1$ and thus require special treatment in the calculation of $k$-size of clusters. Figure 6 shows how this can be handled by noting that whenever the scale function becomes too steep, no cluster can have more than a single sample in any case. That means that we can replace the left and right tails of the scale function with a linear scale without changing the behavior of the algorithm. The

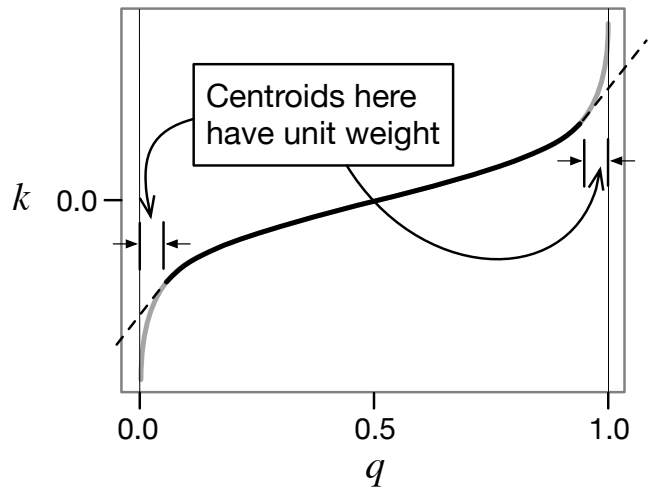


FIGURE 6. When the slope becomes too steep, centroids have unit weight. The effect is that the scaling function (black and gray thick line) is effectively modified (dashed lines) so that the slope is at most $n$, the total number of samples.

2.11. **Size.** End of edited text ... text beyond here needs revision to match current release

2.12. **Accuracy Considerations.** Initial versions of this algorithm tried to use the centroid index $i$ as a surrogate for $q$, applying a correction to account for the fact that extreme centroids have less weight. Unfortunately, it was difficult to account for the fact that the distribution of centroid weights changes during the course of running the algorithm. Initially all weights are 1. Later, some weights become substantially larger. This means that the relationship between $i$ and $q$ changes from completely linear to highly non-linear in a stochastic way. This made it difficult to avoid too large or too small cutoff for centroids resulting in either too many centroids or poor accuracy.

The key property of this algorithm is that the final list of centroids $C$ is very close to what would have been obtained by simply sorting and then grouping adjacent values in $X$ into groups with the desired size bounds. Clearly, such groups could be used to compute all of the rank statistics of interest here and if there are bounds on the sizes of the groups, then we have comparable bounds on the accuracy of the rank statistics in question.

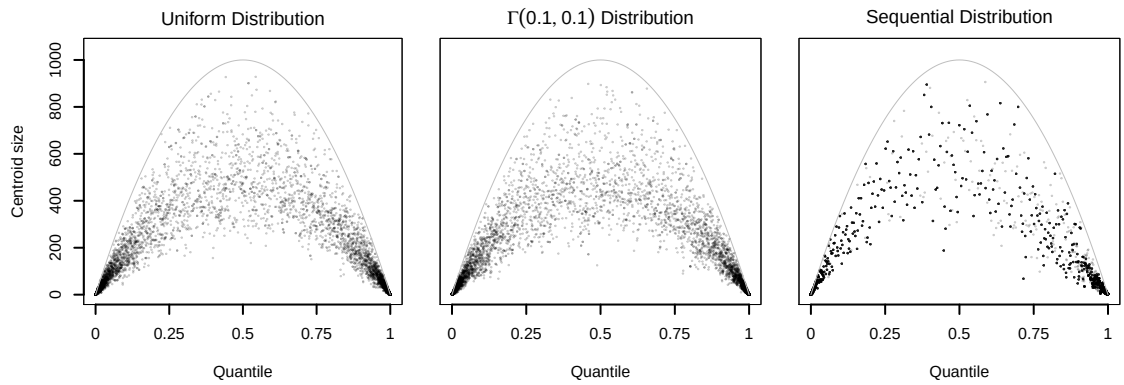


FIGURE 7. The $t$-digest algorithm respects the size limit for centroids. The solid grey line indicates the size limit. These diagrams also shows actual centroid weights for 5 test runs on $100,000$ samples from a uniform, $\Gamma(0.1, 0.1)$ and sequential uniform distribution. In spite of the underlying distribution being skewed by roughly 30 orders of magnitude of difference in probability density for the $\Gamma$ distribution, the centroid weight distribution is bounded and symmetric as intended. For the sequential uniform case, values are produced in sequential order with three passes through the $[0, 1]$ interval with no repeated values. In spite of the lack of repeats, successive passes result in many centroids at the quantiles with the same sizes. In spite of this, sequential presentation of data results in only a small asymmetry in the resulting size distribution and no violation of the intended size limit.

That this algorithm does produce such an approximation is more difficult to prove rigorously, but an empirical examination is enlightening. Figure 8 shows the deviation of samples assigned to centroids for uniform and highly skewed distributions. These deviations are normalized by the half the distance between the adjacent two centroids. This relatively uniform distribution for deviations among the samples assigned to a centroid is found for uniformly distributed samples as well as for extremely skewed data. For instance, the $\Gamma(0.1, 0.1)$ distribution has a 0.01 quantile of $6.07 \times 10^{-20}$, a median of 0.006 and a mean of 1. This means that the distribution is very skewed. In spite of this, samples assigned to centroids near the first percentile are not noticeably skewed within the centroid. The impact of this uniform distribution is that linear interpolation allows accuracy considerably better than $q(1 - q)/\delta$.

2.13. **Finding the cumulative distribution at a point.** Algorithm 3 shows how to compute the cumulative distribution $P(x) = \int_{-\infty}^{x} p(\alpha)\, d\alpha$ for a given value of $x$ by summing the contribution of uniform distributions centered at each the centroids. Each of the

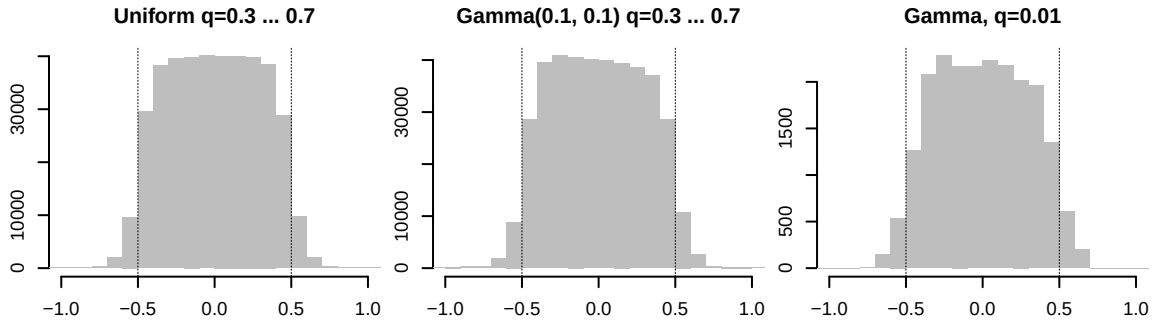


FIGURE 8. The deviation of samples assigned to a single centroid. The horizontal axis is scaled to the distance to the adjacent centroid so a value of 0.5 is half-way between the two centroids. There are two significant observations to be had here. The first is that relatively few points are assigned to a centroid that are beyond the midpoint to the next cluster. This bears on the accuracy of this algorithm. The second observation is that samples are distributed relatively uniformly between the boundaries of the cell. This affects the interpolation method to be chosen when working with quantiles. The three graphs show, respectively, centroids from $q \in [0.3, 0.7]$ from a uniform distribution, centroids from the same range of a highly skewed $\Gamma(0.1, 0.1)$ and centroids from $q \in [0.01, 0.015]$ in a $\Gamma(0.1, 0.1)$ distribution. This last range is in a region where skew on the order of $10^{22}$ is found.

centroids is assumed to extend symmetrically around the mean for half the distance to the adjacent centroid.

For all centroids except one, this contribution will be either 0 or $c_i.\texttt{count}/N$ and the one centroid which straddles the desired value of $x$ will have a *pro rata* contribution somewhere between 0 and $c_i.\texttt{count}/N$. Moreover, since each centroid has count at most $\delta N$ the accuracy of $q$ should be accurate on a scale of $\delta$. Typically, the accuracy will be even better due to the interpolation scheme used in the algorithm and because the largest centroids are only for values of $q$ near 0.5.

The empirical justification for using a uniform distribution for each centroid can be seen by referring to again to Figure 8.

2.14. **Inverting the cumulative distribution.** Computing an approximation of the $q$ quantile of the data points seen so far can be done by ordering the centroids by ascending mean. The running sum of the centroid counts will range from 0 to $N = \sum c_i.\texttt{count}$. One

particular centroid will have a count that straddles the desired quantile $q$ and interpolation can be used to estimate a value of $x$. This is shown in Algorithm 4. Note that at the extreme ends of the distribution as observed, each centroid will represent a single sample so maximum resolution in $q$ will be retained.

2.15. **Computing the trimmed mean.** The trimmed mean of $X$ for the quantile range $Q = [q_0, q_1]$ can be computed by computing a weighted average of the means of centroids that have quantiles in $Q$. For centroids at the edge of $Q$, a *pro rata* weight is used that is based on an interpolated estimate of the fraction of the centroid's samples that are in $Q$. This method is shown as Algorithm 5.

## 3. Empirical Assessment

3.1. **Accuracy of estimation for uniform and skewed distributions.** Figure 9 shows the error levels achieved with $t$-digest in estimating quantiles of 100,000 samples from a uniform and from a skewed distribution. In these experiments $\delta = 0.01$ was used since it provides a good compromise between accuracy and space. There is no visible difference in accuracy between the two underlying distributions in spite of the fact that the underlying densities differ by more roughly 30 orders of magnitude. The accuracy shown here is computed by comparing the computed quantiles to the actual empirical quantiles for the sample used for testing and is shown in terms of $q$ rather than the underlying sample value.

**Algorithm 3:** Estimate quantile $C.\texttt{quantile}(x)$

**Input**: Centroids derived from distribution $p(x)$, $C = [\ldots[m_i, s_i, k_i]\ldots]$ , value $x$
**Output**: Estimated value of $q = \int_{-\infty}^{x} p(\alpha)d\alpha$

```
t = 0, N = Σ_i k_i;
for i ∈ 1...m :
    if i < m :
        Δ ← (c_{i+1}.mean − c_i.mean)/2;
    else:
        Δ ← (c_i.mean − c_{i−1}.mean)/2;
    z = max(−1, (x − m_i)/Δ);
    if z < 1 :
        return (t/N + (k_i/N)(z+1)/2)
    t ← t + k_i;
return 1
```

At extreme values of $q$, the actual samples are preserved as centroids with weight 1 so the observed for these extreme values is zero relative to the original data. For the data shown here, at $q = 0.001$, the maximum weight on a centroid is just above 4 and centroids in this range have all possible weights from 1 to 4. Errors are limited to, not surprisingly, just a few parts per million or less. For more extreme quantiles, the centroids will have fewer samples and the results will typically be exact.

Obviously, with the relatively small numbers of samples such as are used in these experiments, the accuracy of $t$-digests for estimating quantiles of the underlying distribution cannot be better than the accuracy of these estimates computed using the sample data points themselves. For the experiments here, the errors due to sampling completely dominate the errors introduced by $t$-digests, especially at extreme values of $q$. For much larger sample sets of billions of samples or more, this would be less true and the errors shown here would represent the accuracy of approximating the underlying distribution.

It should be noted that using a Q-Digest implemented with long integers is only able to store data with no more than 20 significant decimal figures. The implementation in stream-lib only retains 48 bits of significance, allowing only about 16 significant figures.

**Algorithm 4:** Estimate value at given quantile $C.\texttt{icdf}(q)$

**Input**: Centroids derived from distribution $p(x)$, $C = [c_1 \dots c_m]$ , value $q$
**Output**: Estimated value $x$ such that $q = \int_{-\infty}^{x} p(\alpha) d\alpha$

```
t = 0, q ← q Σ c_i.count;
for i ∈ 1 ... m :
    k_i = c_i.count;
    m_i = c_i.mean;
    if q < t + k_i :
        if i = 1 :
            Δ ← (c_{i+1}.mean − c_i.mean);
        elif i = m :
            Δ ← (c_i.mean − c_{i−1}.mean);
        else:
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        return m_i + ((q−t)/k_i − 1/2) Δ
    t ← t + k_i
return c_m.mean
```

This means that such a Q-digest would be inherently unable to even estimate the quantiles of the $\Gamma$ distribution tested here.

3.2. **Persisting $t$-digests.** For the accuracy setting and test data used in these experiments, the $t$-digest contained $820-860$ centroids. The results of $t$-digest can thus be stored by storing this many centroid means and weights. If centroids are kept as double precision

**Algorithm 5:** Estimate trimmed mean. Note how centroids at the boundary are included on a *pro rata* basis.

**Input**: Centroids derived from distribution $p(x)$, $C = [\ldots[m_i, s_i, k_i]\ldots]$ , limit values $q_0, q_2$

**Output**: Estimate of mean of values $x \in [q_0, q_1]$

```
s = 0, k = 0;
t = 0, q_1 ← q_1 ∑ k_i, q_1 ← q_1 ∑ k_i;
for i ∈ 1 ... m :
    k_i = c_i.count;
    if q_1 < t + k_i :
        if i > 1 :
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        elif i < m :
            Δ ← (c_{i+1}.mean − c_i.mean);
        else:
            Δ ← (c_i.mean − c_{i−1}.mean);
        η = ((q−t)/k_i − 1/2) Δ;
        s ← s + η k_i c_i.mean;
        k ← k + η k_i;
    if q_2 < t + k_i :
        if i > 1 :
            Δ ← (c_{i+1}.mean − c_{i−1}.mean)/2;
        elif i < m :
            Δ ← (c_{i+1}.mean − c_i.mean);
        else:
            Δ ← (c_i.mean − c_{i−1}.mean);
        η = (1/2 − (q−t)/k_i) Δ;
        s ← s − η k_i c_i.mean;
        k ← k − η k_i;
    t ← t + k_i
return s/k
```

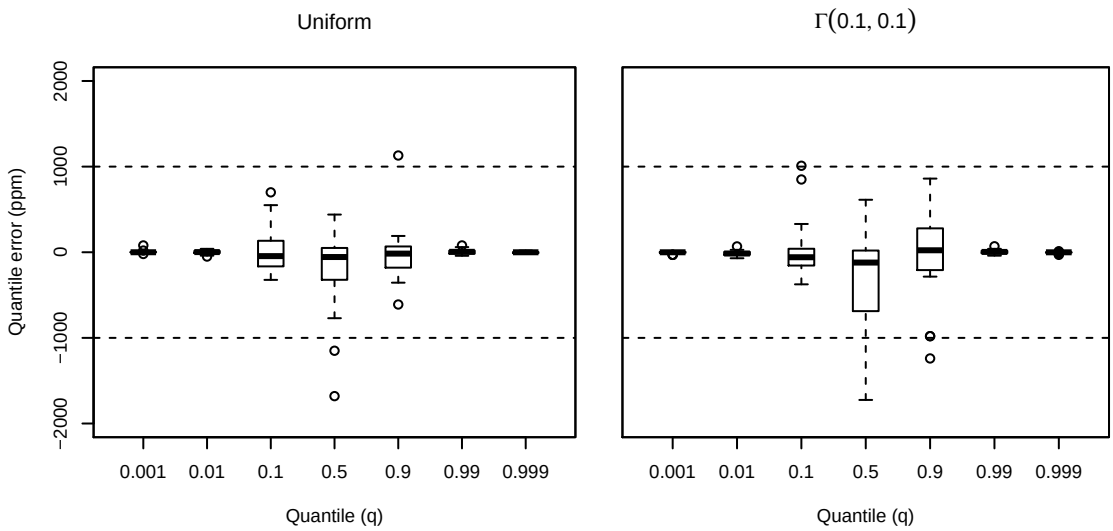


FIGURE 9. The absolute error of the estimate of the cumulative distribution function $q = \int_{-\infty}^{x} p(\alpha)\, d\alpha$ for the uniform and $\Gamma$ distribution for 5 runs, each with 100,000 data points. As can be seen, the error is dramatically decreased for very high or very low quantiles (to a few parts per million). The precision setting used here, $1/\delta = 100$, would result in uniform error of 10,000 ppm without adaptive bin sizing and interpolation.

floating point numbers and counts kept as 4-byte integers, the $t$-digest resulting from from the accuracy tests described here would require about 10 kilobytes of storage for any of the distributions tested.

This size can be substantially decreased, however. One simple option is to store differences between centroid means and to use a variable byte encoding such as zig-zag encoding to store the cluster size. The differences between successive means are at least three orders of magnitude smaller than the means themselves so using single precision floating point to store these differences can allow the $t$-digest from the tests described here to be stored in about 4.6 kilobytes while still regaining nearly 10 significant figures of accuracy in the means. This is roughly equivalent to the precision possible with a Q-digest operating on 32 bit integers, but the dynamic range of $t$-digests will be considerably higher and the accuracy is considerably better.

3.3. **Space/Accuracy Trade-off.** Not surprisingly, there is a strong trade-off between the size of the $t$-digest as controlled by the compression parameter $\delta$ and the accuracy

which which quantiles are estimated. Quantiles at 0.999 and above or 0.001 or below were estimated to within a small fraction of 1% regardless of digest size. Accurate estimates of the median require substantially larger digests. Figure 10 shows this basic trade-off.

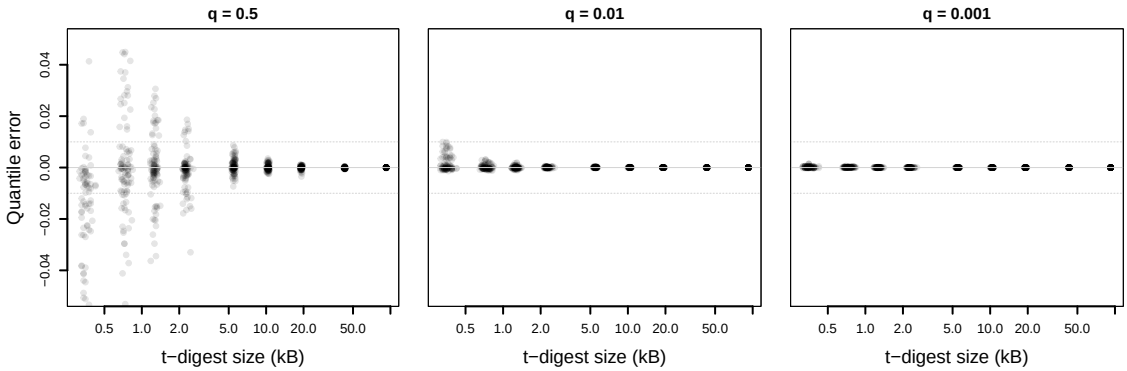


FIGURE 10. Accuracy is good for extreme quantiles regardless of digest size. From left to right, these panels show accuracy of estimates for $q = 0.5, 0.01$ and 0.001 as a function the serialized size of the $t$-digest. Due to symmetry, these are equivalent to accuracy for $q = 0.5, 0.99$, and 0.999 as well. For mid quantiles such as the median ($q = 0.5$), moderate digest sizes of a few kilobytes suffice to get better than 1% accuracy, but a digest must be 20kB or more to reliably achieve 0.1% accuracy. In contrast, accuracy for the 0.1%-ile (or 99.9%-ile) reaches a few parts per million for digests larger than about 5 kB. Note that errors for $q = 0.5$ and digests smaller than 1 kB are off the scale shown here at nearly 10%. All panels were computed using 100 runs with 100,000 samples. Compression parameter ($1/\delta$) was varied from 2 to 1000 in order to vary the size of the resulting digest. Sizes shown were encoded using 4 byte floating point delta encoding for the centroid means and variable byte length integer encoding.

The size of the resulting digest depends strongly on the compression parameter $1/\delta$ as shown in the left panel of Figure 11. Size of the digest also grows roughly with the log of the number of samples observed, at least in the range of 10,000 to 10,000,000 samples shown in the right panel of Figure 11.

3.4. **Computing $t$-digests in parallel.** With large scale computations, it is important to be able to compute aggregates like the $t$-digest on portions of the input and then combine those aggregates.

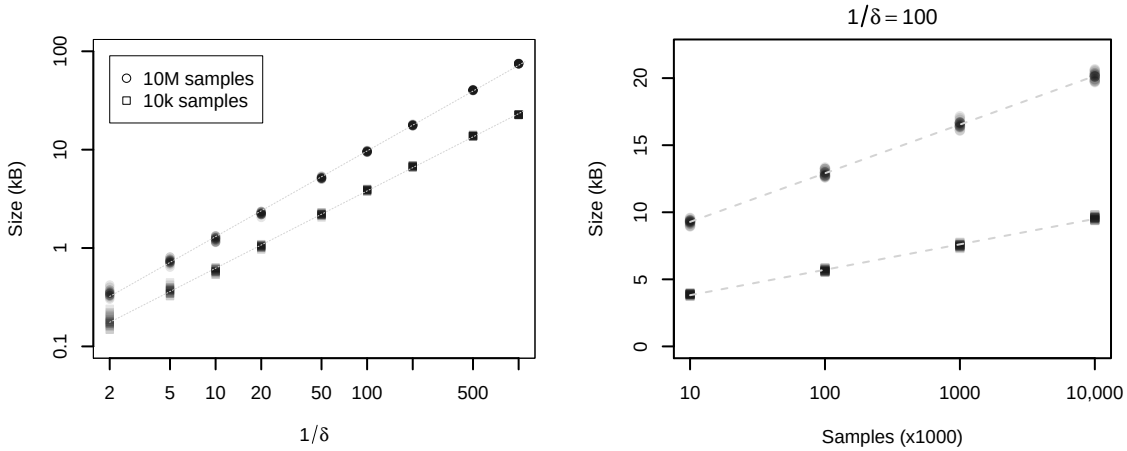


FIGURE 11. Size of the digest scales sub-linearly with compression parameter ($\alpha \approx 0.7 \ldots 0.9$) for fixed number of points. Size scales approximately logarithmically with number of points for fixed compression parameter. The panel on the right is for $1/\delta = 100$. The dashed lines show best-fit log-linear models. In addition, the right panel shows the memory size required for the GK01 algorithm if 6 specific quantiles are desired.

For example, in a map-reduce framework such as Hadoop, a combiner function can compute the $t$-digest for the output of each mapper and a single reducer can be used to compute the $t$-digest for the entire data set.

Another example can be found in certain databases such as Couchbase or Druid which maintain tree structured indices and allow the programmer to specify that particular aggregates of the data being stored can be kept at interior nodes of the index. The benefit of this is that aggregation can be done almost instantaneously over any contiguous sub-range of the index. The cost is quite modest with only a $O(\log(N))$ total increase in effort over keeping a running aggregate of all data. In many practical cases, the tree can contain only two or three levels and still provide fast enough response. For instance, if it is desired to be able to compute quantiles for any period up to years in 30 second increments, simply keeping higher level $t$-digests at the level of 30 seconds and days is likely to be satisfactory because at most about 10,000 digests are likely to need merging even for particularly odd intervals. If almost all queries over intervals longer than a few weeks are day aligned, the number of digests that must be merged drops to a few thousand.

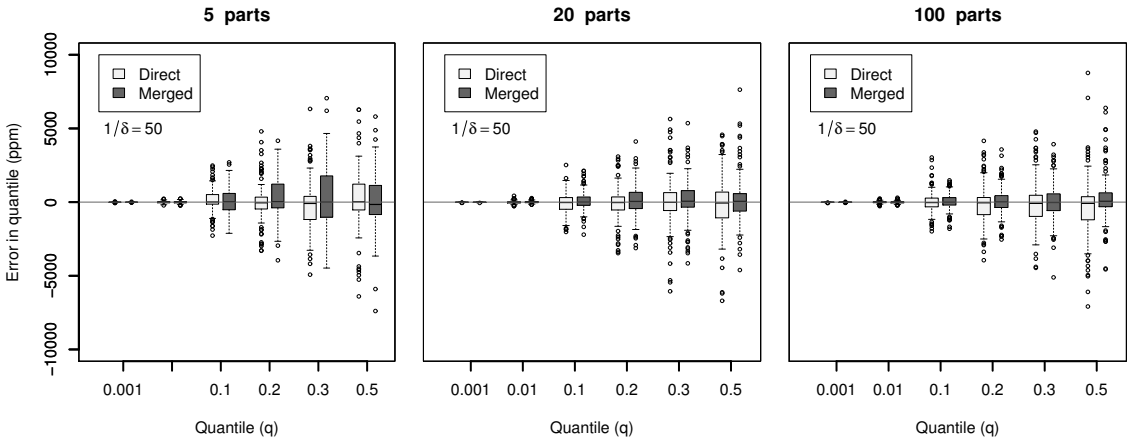


FIGURE 12. Accuracy of a $t$-digest accumulated directly is nearly the same as when the digest is computed by combining digests from 20 or 100 equal sized portions of the data. Repeated runs of this test occasionally show the situation seen in the left panel where the accuracy for digests formed from 5 partial digests show slightly worse accuracy than the non-subdivided case. This sub-aggregation property allows efficient use of the $t$digest in map-reduce and database applications. Of particular interest is the fact that accuracy actually improves when the input data is broken in to many parts as is shown in the right hand panel. All panels were computed by 40 repetitions of aggregating 100,000 values. Accuracy for directly accumulated digests is shown on the left of each pair with the white bar and the digest of digest accuracy is shown on the right of each pair by the dark gray bar.

Merging $t$-digests can be done many ways. The algorithm whose results are shown here consisted of simply making a list of all centroids from the $t$-digests being merged, shuffling that list, and then adding these centroids, preserving their weights, to a new $t$-digest.

3.5. **Comparison with Q-digest.** The prototype implementation of the $t$-digest completely dominates the implementation of the Q-digest from the popular stream-lib package [Thi] when size of the resulting digest is controlled. This is shown in Figure 13. In the left panel, the relationship between the effect of the compression parameter for Q-digest is compared to the similar parameter $\delta$ for the $t$-digest. For the same value of compression parameter, the sizes of the two digests is always within a factor of 2 for practical uses. The middle and right panel show accuracies for uniform and $\Gamma$ distributions.

As expected, the $t$-digest has very good accuracy for extreme quantiles while the Q-digest has constant error across the range. Interestingly, the accuracy of the Q-digest is at

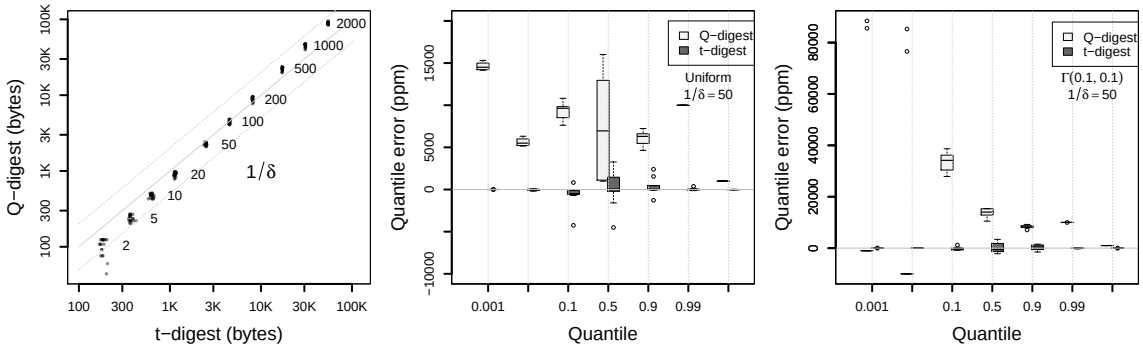


FIGURE 13. The left panel shows the size of a serialized Q-digest versus the size of a serialized t-digest for various values of $1/\delta$ from 2 to 100,000. The sizes for the two kinds of digest are within a factor of 2 for all compression levels. The middle and right panels show the accuracy for a particular setting of $1/\delta$ for Q-digest and t-digest. For each quantile, the Q-digest accuracy is shown as the left bar in a pair and the t-digest accuracy is shown as the right bar in a pair. Note that the vertical scale in these diagrams are one or two orders of magnitude larger than in the previous accuracy graphs and that in all cases, the accuracy of the t-digest is dramatically better than that of the Q-digest even though the serialized size of the each is within 10% of the other. Note that the errors in the right hand panel are systematic quantization errors introduced by the use of integers in the Q-digest algorithm. Any distribution with very large dynamic range will show the same problems.

best roughly an order of magnitude worse than the accuracy of the $t$-digest even. At worse, with extreme values of $q$, accuracy is several orders of magnitude worse. This situation is even worse with a highly skewed distribution such as with the $\Gamma(0.1, 0.1)$ shown in the right panel. Here, the very high dynamic range introduces severe quantization errors into the results. This quantization is inherent in the use of integers in the Q-digest.

For higher compression parameter values, the size of the Q-digest becomes up to two times smaller than the $t$-digest, but no improvement in the error rates is observed.

3.6. **Speed.** The current implementation has been primarily optimized for ease of development, not execution speed. As it is, running on a single core of a 2.3 GHz Intel Core i5, it typically takes 2-3 microseconds to process each point after JIT optimization has come into effect. It is to be expected that a substantial improvement in speed could be had by profiling and cleaning up the prototype code.

## 4. Conclusion

The $t$-digest is a novel algorithm that dominates the previously state-of-the-art Q-digest in terms of accuracy and size. The $t$-digest provides accurate on-line estimates of a variety of of rank-based statistics including quantiles and trimmed mean. The algorithm is simple and empirically demonstrated to exhibit accuracy as predicted on theoretical grounds. It is also suitable for parallel applications. Moreover, the $t$-digest algorithm is inherently on-line while the Q-digest is an on-line adaptation of an off-line algorithm.

The $t$-digest algorithm is available in the form of an open source, well-tested implementation from the author. It has already been adopted by the Apache Mahout and stream-lib projects and is likely to be adopted by other projects as well.

## References

[CLP00] Fei Chen, Diane Lambert, and Jos C. Pinheiro. Incremental quantile estimation for massive tracking. In *In Proceedings of KDD*, pages 516–522, 2000.

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *In SIGMOD*, pages 58–66, 2001.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*, page 232. Addison-Wesley, Boston, 3 edition, 1998.

[Lin] LinkedIn. Datafu: Hadoop library for large-scale data processing. `https://github.com/linkedin/datafu/`. [Online; accessed 20-December-2013].

[MP80] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315 – 323, 1980.

[PDGQ05] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Sci. Program.*, 13(4):277–298, October 2005.

[SBAS04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: New aggregation techniques for sensor networks. pages 239–249. ACM Press, 2004.

[Thi] Add This. Algorithms for calculating variance, online algorithm. `https://github.com/addthis/stream-lib`. [Online; accessed 28-November-2013].

[Wel62] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, pages 419–420, 1962.

[Wik] Wikipedia. Algorithms for calculating variance, online algorithm. `http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Online_algorithm`. [Online; accessed 19-October-2013].

Ted Dunning, MapR Technologies, Inc, San Jose, CA

*E-mail address*: ted.dunning@gmail.com

*E-mail address*: otmar.ertl@gmail.com