STOCHASTIC GRADIENT DESCENT (SGD) FOR QUANTILE ESTIMATION

YIPING SU

Supervisor: Cheng Soon Ong
Bachelor of Information Technology (Honours)
College of Engineering and Computer Science
Australian National University

June 2020

ABSTRACT

Finding the quantiles of an unknown distribution is a good way of characterising the distribution, but for large amounts of streaming data, this is infeasible. Instead, methods for estimation of quantiles of streaming data have been developed for decades but are rarely related with the rapidly growing topic machine learning. Stochastic gradient descent is a common machine learning method, which we apply in our quantile estimation algorithm SGD. This simple SGD algorithm is shown to be equivalent in some sense to the state-of-the-art Frugal-1U. In an effort to understand the convergence of SGD, we empirically compare the SGD performance under different settings of data distribution, data size, data ordering and SGD step size. Our experiments show that SGD is sensitive to the distribution and size of the data, as well as the step size. As the only such parameter we can control, we propose two step size adaptation approaches, which improve the convergence rate of SGD on our test data. Finally, we explore two algorithms for simultaneous estimation of multiple quantiles, and briefly discuss how their optimisations might be applied to our SGD algorithm. Despite its simplicity, SGD converges to the true quantile with only O(1) space complexity, making it a very efficient quantile estimation algorithm.

CONTENTS

1 INTRODUCTION AND BACKGROUND 1
1.1 Quantiles . . . 1
1.2 Data streams and quantile estimation . . . 2
1.3 Stochastic gradient descent(SGD) for quantile estimation . . . 3
1.4 Background . . . 3
1.4.1 Gradient descent and stochastic gradient descent . . . 3
1.4.2 Quantile . . . 5
1.5 Thesis overview . . . 6
2 LITERATURE REVIEW 9
2.1 Quantile estimation methods on streaming data . . . 9
2.1.1 Deterministic algorithms . . . 10
2.1.2 Randomised algorithms . . . 10
2.1.3 Algorithms for other requirements . . . 11
2.2 Non-growing space for single quantile estimation . . . 12
2.3 Non-growing space for multi-quantile estimation . . . 13
2.3.1 Other methods and advantages of simultaneous estimation 14
3 EQUIVALENCE OF SGD AND FRUGAL-1U 15
3.1 Pinball Loss for Quantile Estimation . . . 15
3.2 Deriving the SGD approach from Frugal . . . 16
3.2.1 Pseudo Code for the Frugal-1U Algorithm . . . 16
3.2.2 SGD for pinball loss function . . . 17
3.3 Equivalence of Algorithms . . . 18
3.3.1 Comparison experiments between SGD and Frugal-1U . . . 18
3.3.2 Discussion about the equivalence . . . 20
3.4 Conclusion . . . 20
4 EMPIRICAL INVESTIGATION OF SGD CONVERGENCE 25
4.1 Methodology . . . 25
4.1.1 Benchmarking Estimate Setting . . . 25
4.1.2 SGD Quantile Estimate Generation . . . 26
4.2 Evaluation methods . . . 27
4.2.1 Error Computation . . . 28
4.2.2 Performance Comparison . . . 29
4.3 Experiment Observations . . . 30
4.3.1 Distribution . . . 30
4.3.2 Data Size . . . 37
4.3.3 Ordering of Data Stream . . . 42
4.3.4 SGD Step Size . . . 43
4.4 Discussion . . . 48
4.4.1 SGD step sizes and distributions . . . 48
4.4.2 SGD step size and data size . . . 48
4.4.3 Starting point and the convergence . . . 48
4.5 Conclusion . . . 49
5 STEP SIZE ADAPTATION 51
5.1 Failure on Newton's Method . . . 51
5.2 Stochastic Average Gradient (SAG) . . . 52

5.2.1 Mechanism of SAG . . . 52
5.2.2 Basic SAG algorithm . . . 53
5.2.3 SAG Implementation for quantile estimation . . . 53
5.2.4 Experiments on SAG . . . 54
5.2.5 Smooth Functions . . . 56
5.3 Doubling and Halving SGD (DH-SGD) . . . 59
5.3.1 Method Description . . . 59
5.3.2 Experiments . . . 61
5.3.3 Improvements on DH-SGD . . . 62
5.4 Conclusion . . . 63
6 SIMULTANEOUS MULTIPLE QUANTILE ESTIMATION 65
6.1 The problem and opportunity of multi-quantile estimation . . . 65
6.1.1 The crossing problem of SGD . . . 66
6.2 The shiftQ method . . . 67
6.2.1 Method Description . . . 68
6.2.2 Experiment Results . . . 69
6.3 The Extended P^2 Algorithm . . . 73
6.3.1 Method Description . . . 73
6.3.2 Experiment Results . . . 75
6.4 Discussion and Conclusion . . . 78
7 CONCLUSION 81
7.1 Future work . . . 81
7.2 Summary . . . 82

BIBLIOGRAPHY 83

LIST OF FIGURES

Figure 1 Quantiles (0.5-q, 0.9-q and 0.99-q) of a dataset containing 2000 random samples from a Gaussian distribution (mean = 2, standard deviation = 18) 1

Figure 2 SGD quantile estimation of the 0.99-q for a dataset of 2000 samples from a Gaussian distribution. The left graph is a combination of incoming data points and the SGD steps, and each step of SGD is triggered by a new coming data point. The blue line shows how the SGD result is updated on the arrival of a data point (sea-green), and straight line (violet) represents the empirical value of 0.99-q. On the right side, the density of the bell-shaped dataset is shown in a histogram. 3

Figure 3 The relationship between topics covered in the thesis. Topics are roughly positioned along the top-bottom axis depending on where they are more close to SGD methods (left) or non-SGD methods (right). The arrows between the chapters represent are connected according to dependence. 7

Figure 4 Quantile estimation behaviour similarity between Frugal-1U and SGD on the *gau-1* dataset. The horizontal dotted lines are the true quantiles, the dark solid colour lines are SGD quantiles, and the lighter solid colour lines are the mean value of Frugal-1U quantiles. The light coloured shadows around the solid lines are the range of Frugal-1U steps over the 100 runs. For each epoch, the top and the bottom margin of the shadow are positioned respectively on the maximum and minimum value of the 100 experiments. The shadow of a quantile is an indicator of how much the Frugal-1U estimate varies around the mean value. Unfortunately the shadows from different quantiles sometimes cover each other, so the it can only be used as a rough visual reference of changes in Frugal-1U. 20

Figure 5 Quantile estimation result similarity between Frugal-1U and SGD on the *gau-1* dataset. Each of the 5 histograms represents the distribution of Frugal-1U similarity error values for a specific quantile. The similarity error values are compared by function (18). For example, consider the top one (the 0.1-quantile). The distribution of 100 similarity error values are shown in the bright blue histogram. The mean similarity error value is also in the form of the vertical black line. 21

Figure 6 Quantile estimation behaviour similarity between Frugal-1U and SGD on the *mix* dataset 21

Figure 7 Quantile estimation result similarity between Frugal-1U and SGD on the *mix* dataset 22

Figure 8 Quantile estimation behaviour similarity between Frugal-1U and SGD on the *exp* dataset . . . 22
Figure 9 Quantile estimation result similarity between Frugal-1U and SGD on the *exp* dataset . . . 23
Figure 10 SGD Process from *gau-1* Distribution . . . 31
Figure 11 SGD Process from *gau-2* Distribution . . . 32
Figure 12 SGD Process from *mix* Distribution . . . 32
Figure 13 SGD Process from *exp* Distribution . . . 33
Figure 14 SGD Results from *gau-1* Distribution . . . 33
Figure 15 SGD Results from *gau-2* Distribution . . . 33
Figure 16 SGD Results from *mix* Distribution . . . 34
Figure 17 SGD Results from *exp* Distribution . . . 34
Figure 18 SGD Error from *gau-1* Distribution . . . 34
Figure 19 SGD Error from *gau-2* Distribution . . . 35
Figure 20 SGD Error from *mix* Distribution . . . 35
Figure 21 SGD Error from *exp* Distribution . . . 36
Figure 22 SGD Process from 100 Samples . . . 38
Figure 23 SGD Process from 1000 Samples . . . 38
Figure 24 SGD Process from 100000 Samples . . . 39
Figure 25 SGD Results from 100 Samples . . . 39
Figure 26 SGD Results from 1000 Samples . . . 39
Figure 27 SGD Results from 100000 Samples . . . 40
Figure 28 SGD Error from 100 Samples . . . 40
Figure 29 SGD Error from 1000 Samples . . . 40
Figure 30 SGD Error from 100000 Samples . . . 41
Figure 31 SGD Estimate Process from Shuffled Data Stream . . . 42
Figure 32 SGD Estimate Result from Shuffled Data Stream . . . 43
Figure 33 SGD Estimate Error from Shuffled Data Stream . . . 43
Figure 34 SGD Estimate Process from step size $\alpha_i = 1$ . . . 44
Figure 35 SGD Estimate Process from step size $\alpha_i = \frac{2}{\sqrt{i}}$ . . . 44
Figure 36 SGD Estimate Process from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$ . . . 45
Figure 37 SGD Estimate Results from step size $\alpha_i = 1$ . . . 45
Figure 38 SGD Estimate Results from step size $\alpha_i = \frac{2}{\sqrt{i}}$ . . . 45
Figure 39 SGD Estimate Results from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$ . . . 46
Figure 40 SGD Estimate Error from step size $\alpha_i = 1$ . . . 46
Figure 41 SGD Estimate Error from step size $\alpha_i = \frac{2}{\sqrt{i}}$ . . . 46
Figure 42 SGD Estimate Error from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$ . . . 47
Figure 43 SAG Process from *gau-1* Distribution . . . 55
Figure 44 SAG Result from *gau-1* Distribution . . . 55
Figure 45 SAG Error from *gau-1* Distribution . . . 55
Figure 46 Comparison between $|x|$ and the pinball loss function with different τ values . . . 56
Figure 47 Comparison between the two smooth functions when $\mu = 0.001$ . . . 58
Figure 48 Comparison between the two smooth functions when $\mu = 0.0001$ . . . 58
Figure 49 DH-SGD Process from *gau-1* Distribution . . . 61
Figure 50 DH-SGD Results from *gau-1* Distribution . . . 62
Figure 51 DH-SGD Error from *gau-1* Distribution . . . 62

Figure 52 DH-SGD Error from *gau-1* Distribution 63
Figure 53 The update of multi-quantile estimation methods in general 65
Figure 54 Three process plots comparing the estimation of quantiles 0.15-q, 0.16-q, 0.17-q, 0.25-q, 0.8-q and 0.9-q with SGD over a uniform distribution. The step size of the SGD algorithms are 5, 1, and 0.45. In this experiment, each of them respectively has 2882, 659 and 33 epochs with crossing quantiles. 66
Figure 55 Two process plots comparing the estimation of quantiles 0.15-q, 0.16-q, 0.17-q, 0.25-q, 0.8-q and 0.9-q with shiftQ and SGD over a uniform distribution. 70
Figure 56 The shiftQ algorithm for Positive Gaussian 1 distribution (the process graph) 70
Figure 57 The shiftQ algorithm for Positive Gaussian 1 distribution (the result graph) 71
Figure 58 The shiftQ algorithm for Positive Gaussian 1 distribution (the error graph) 71
Figure 59 The shiftQ algorithm for Positive Gaussian 1 distribution (the process graph) 71
Figure 60 Relationship between τ value, marker position and quantile values correspondingly for 3 adjacent quantiles 73
Figure 61 Quantile value update using the Piecewise-Parabolic(P^2) formula 74
Figure 62 The extended P^2 algorithm for Gaussian 1 distribution (the process graph) 77
Figure 63 The extended P^2 algorithm for Gaussian 1 distribution (the result graph) 77
Figure 64 The extended P^2 algorithm for Gaussian 1 distribution (the error graph) 78

1 INTRODUCTION AND BACKGROUND

In the situation when a large amount of data comes like a flow, data analysts might be interested in the how is the data distributed, or in other words, where are the densely and sparse parts of the distribution. For example, the managers of an online shopping system might be interested in the distribution of customers' age of an online sales event. As they want to find the majority of age groups for a more efficient advertisement, the question of the distribution is "what is the age diving points of the youngest 10% and the oldest 10%". On one hand, the most updated distribution analysis is expected for a rapid adaptation, while on the other hand there is limited storage for the large amount of incoming data. In this paper, we explore this problem as a quantile estimation problem on data streams, and try to find space-efficient and computationally cheap solutions using the stochastic gradient descent method.

1.1 QUANTILES

Quantiles are the cutting points of a statistical distribution by its ranking. Specifically, the τ-quantile is the cutting point that divides the distribution by probability τ. Let τ-q denote the τ-quantile. For example, a 0.5-q is the median of a distribution, such that there is a 50% probability that a random sample is smaller than it. Generally, the value of a quantile can be achieved by mathematical approaches when the expression of the distribution is given. The quantiles are an important statistical feature of distributions for their values roughly reflects the density of the distribution. For example, Fig 1 shows samples from a Gaussian distribution and is annotated with two intervals [0.5-q, 0.9-q] and [0.9-q, 0.99-q]. The values of the intervals are similar, but the former contains 40% data and the latter contains only 9% data. Without any other information, this alone leads to the implication that the distribution is denser in the former interval than the latter.

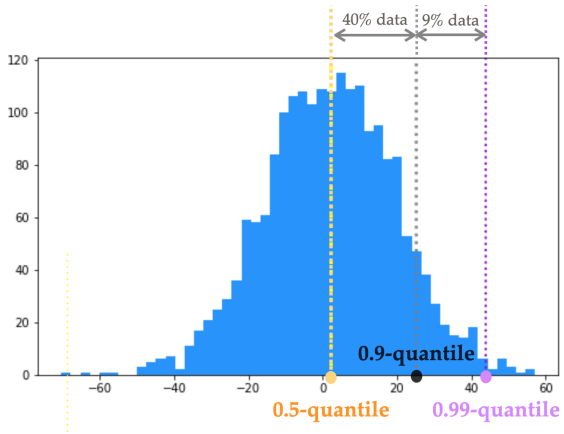

Figure 1: Quantiles (0.5-q, 0.9-q and 0.99-q) of a dataset containing 2000 random samples from a Gaussian distribution (mean = 2, standard deviation = 18)

As shown in the example of Fig 1, the concept of quantiles are also applied in datasets as well as probability distributions. Similarly, the quantile for a finite dataset is the point which divides the dataset by probability τ. However, since the dataset is discrete, the value of a τ-q is ambiguous. For example, for dataset $[1, 2, 3, 4]$, both 2.01 and 2.99 can be considered as a valid value of 0.5-q. Among various methods of dataset quantile estimations, we apply the one currently used by the *Python 3* language. For a size N dataset $X = \{x_1, ..., x_N\}$, the method first finds the ranking index of the quantile $h = (N-1)\tau + 1$. If h is not an integer, the estimated quantile is computed by linear interpolation between the two data points at ranking positions surround h

$$\tau\text{-}q_{batch} = x_{\lfloor h \rfloor} + (h - \lfloor h \rfloor)\left(x_{\lfloor h \rfloor + 1} - x_{\lfloor h \rfloor}\right) \tag{1}$$

where $\lfloor h \rfloor$ is the greatest integer less than or equal to h. The computation result $\tau\text{-}q_{batch}$ is called a *batch quantile*, since it comes from a batch of samples of a distribution.

Note that although computing the quantiles of a sample estimates the quantiles of the associated distribution, this is not q̈uantile estimationäs it is referred to in this paper. Here the batch quantiles are regarded as computed quantiles, as a comparison of *true quantiles* which are the real quantiles of the original probability distribution. The estimation of quantiles is introduced in the following part.

1.2 DATA STREAMS AND QUANTILE ESTIMATION

A *Data stream* is a large source of data where data is created in sequence over a period of time. In contrast with datasets, data points are not instantly available all at a time, and that the size of the sample grows over time. Data streams are commonly seen in areas like network monitoring, data mining, financial trading systems, etc. Similarly to normal datasets, the value of quantiles is important for data analysis of data streams. Finding the quantiles of a data stream is the initial aim of this paper.

A trivial solution to find quantiles of data streams is to sort the entire data stream when the last data point arrives, and then compute the batch quantiles of the sorted dataset. In cases when the size of the data stream is unknown, at the arrival of each data point, the batch quantiles are computed again so the quantile values get updated. The method of repeatedly sorting and computing batch quantiles is called the *batch algorithm*. Due to the large size of the data streams, the batch algorithm for quantile computation is too expensive in both storage and computation to be a feasible solution for most computer systems.

Faced with the storage and computation problem, algorithms of *quantile estimation* on data streams have been proposed. The quantile estimation algorithms do not store the entire data streams, and the algorithms return estimated quantiles that are close to the batch quantiles. Some quantile estimation algorithms are described in the literature review chapter. Although these algorithms use significantly less memory than the batch algorithm, most require a growing space complexity (i.e., correlated with size N). In this paper we investigate the space-efficient algorithms that use constant memory units for data streams of any size. Specifically, we focus on the machine learning method *stochastic gradient descent* (SGD) for quantile estimation.

1.3 STOCHASTIC GRADIENT DESCENT(SGD) FOR QUANTILE ESTIMATION

Gradient descent is a convex optimization algorithm that is commonly used in machine learning for loss function minimization. Gradient descent takes the entire dataset as the input, and by iteratively moving in the opposite direction of the gradient, it reaches the local minimum. Such method can also be used in quantile estimation if the entire dataset is available all at the same time.

Stochastic gradient descent (SGD), on the other hand, is the "online learning" version of gradient descent that updates iteratively when each new data point comes. For each data point, SGD steps in the opposite direction of the gradient computed from that data point. In this way, SGD updates using one data point at a time, rather than the whole dataset. Fig 2 shows how SGD is used to estimate quantiles on the same dataset of 2000 Gaussian distribution samples from Fig 1. It is shown in the figure that the SGD result fluctuates around the empirical quantile value, indicating SGD is a possible approach for quantile estimation.

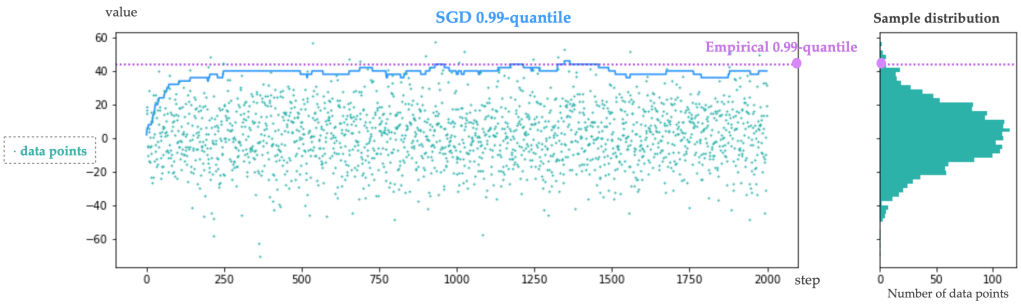

Figure 2: SGD quantile estimation of the 0.99-q for a dataset of 2000 samples from a Gaussian distribution. The left graph is a combination of incoming data points and the SGD steps, and each step of SGD is triggered by a new coming data point. The blue line shows how the SGD result is updated on the arrival of a data point (sea-green), and straight line (violet) represents the empirical value of 0.99-q. On the right side, the density of the bell-shaped dataset is shown in a histogram.

1.4 BACKGROUND

In this chapter, the essential background knowledge assumed throughout the rest of this thesis is defined. Subsection 1.4.1introduces Stochastic Gradient Descent (SGD), which is a convex optimisation approach commonly found in machine learning, and subsection 1.4.2 provides a definition for quantiles. Stochastic gradient descent (SGD) is a convex optimization approach commonly applied in machine learning, and quantiles are the dividing points of a distribution or a dataset.

1.4.1 *Gradient descent and stochastic gradient descent*

Stochastic gradient descent is an optimization algorithm developed from gradient descent.

1.4.1.1 *Gradient Descent*

For convex optimization problems, gradient descent is a first-order optimization algorithm to find the local minimum of a function.

To solve the minimization problem

$$\min_{x} L(x) \tag{2}$$

where $L : \mathbb{R}^d \to \mathbb{R}$ is convex, differentiable and its gradient is Lipschitz continuous with constant $L > 0$.

Geometrically, the gradient $\nabla L(x_0)$ points to the direction of the steepest ascent on $L(\cdot)$ from the point x_0. By taking a small step in the direction of the negative gradient, the function value is decreased in the direction of the steepest descent. That is,

$$x_1 = x_0 - \alpha \nabla L(x_0) \tag{3}$$

for a small enough step size $\alpha \in \mathbb{R}_+$, then $L(x_1) \leqslant L(x_0)$. That means, compared with $L(x_0)$, $L(x_1)$ is closer to the local minimum.

With this observation comes the idea of gradient descent: an iterative "tour" on $L(\cdot)$ from a point towards the local minimum by following small steps of negative gradient. Let x_0 be the guess of a starting point, then if

$$x_{i+1} = x_i - \alpha_i \nabla L(x_i), i \geqslant 0 \tag{4}$$

Then we have $L(x_0) \geqslant L(x_1) \geqslant L(x_2) \geqslant \cdots$ where α_i is a suitable step size for iteration i. The convergence rate of the sequence (x_n) with certain step size settings is linear in terms of the number of iterations.

The objective function $L(\cdot)$ can be written as a sum of differentiable functions:

$$L(x) = \frac{1}{N} \sum_{n=1}^{N} \ell_n(x) \tag{5}$$

where the *summand function* ℓ_n is usually the loss function of the nth observation among N data points.

Applying the gradient descent formula, x is updated according to

$$x_{i+1} = x_i - \alpha_i \nabla L(x_i) = x_i - \alpha_i \frac{1}{N} \sum_{n=1}^{N} \nabla \ell_n(x_i) \tag{6}$$

From formula 6, it is clear that gradient descent requires access to all the data points to compute the movement direction, which is infeasible for a data stream.

1.4.1.2 *Stochastic Gradient Descent (SGD)*

It is the choice for data streams, because the computation of the movement direction relies only on the most recent data point. It can be considered as a stochastic approximation of gradient descent optimization.

The calculation of $\sum_{n=1}^{N} \nabla \ell_n(\mathbf{x}_i)$ can be expensive, especially when the amount of summand functions is huge, or when the individual gradients are hard to compute. To reduce the calculation, an estimation of the true gradient of $L(\mathbf{x})$ is taken: the true gradient $\frac{1}{N} \sum_{n=1}^{N} \nabla \ell_n(\mathbf{x}_i)$ is replaced by the gradient with respect to the mth observation, $\nabla \ell_m(\mathbf{x}_i)$. So the update of the parameter $\mathbf{x}$ becomes

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha_i \nabla \ell_m(\mathbf{x}_i) \tag{7}$$

Generally, each time m is randomly selected from N. In this paper, we use $m = i$ for quantile estimation on data streams.

1.4.1.3 *Limitations*

Since each SGD update requires only a single data point instead of the entire dataset, it becomes a great alternative for quantile estimation. However, neither GD nor SGD can handle non-differentiable functions. This is an issue for the quantile estimation loss function in the following chapter. Furthermore, while the computation per update is decreased N times, the tradeoff lies in the convergence rate. SGD does not enjoy the same linear convergence rate as gradient descent. Both of these issues are discussed in chapter 5.

1.4.2 *Quantile*

In statistics, quantiles are the points that divide a probability distribution into even intervals. The q-quantiles ($q \in \{2, 3, 4, ...\}$) is a set of quantile points which divides the distribution into q intervals each with the same probability mass. For example, the 2-quantile has only one quantile point, which is the middle point of the distribution and it divides the distribution into two even parts. This 2-quantile point is called the median.

Definition with q-quantile notation
Generally, the q-quantiles have $q - 1$ quantile points, and the kth quantile point for a distribution X is the data value such that

$$Pr(X \leqslant x) \geqslant \frac{k}{q} \tag{8}$$

and

$$Pr(X \geqslant x) \geqslant 1 - \frac{k}{q} \tag{9}$$

where $x \in X$.

Definition with notation τ
Since $k < q$, we have $0 < \frac{k}{q} < 1$ for all (k,q) pairs. For $k, q \in \mathbb{N}^+$ we can define a quantile for any rational number in the interval $(0, 1)$. For a quantile probability τ, we use the notation τ-quantile ($0 < \tau < 1$) for a quantile value in the distribution X that satisfies

$$Pr(X \leqslant x) \geqslant \tau \tag{10}$$

and

$$Pr(X \geqslant x) \geqslant 1 - \tau \tag{11}$$

where $x \in X$.

1.5 THESIS OVERVIEW

This thesis aims to thoroughly answer the following question:

Can SGD be used for effective quantile estimation?

In investigating the above research question, this thesis includes the following contributions:

1. Proposal of the SGD algorithm for quantile estimation which implements the SGD approach.
2. SGD algorithm performance comparison with the state-of-the-art Frugal1U.
3. Identification of some contributing factors on SGD performance.
4. Improved SGD methods with quicker convergence rate to the true quantile value.
5. Experiments on other people's methods that simultaneously estimate different quantiles at the same time, which is a potential improvement direction for SGD too.

In the final part of the introduction, we give a brief tour of the material in this paper. The main focus of the paper is to explore how stochastic gradient descent method can be used for quantile estimation on data streams. In chapter 2, a very brief discussion of quantile estimation algorithms is presented, with a special mention for space-efficient algorithms (e.g., SGD-like methods). Chapter 3 compares the SGD methods with the *Frugal1U* algorithm[21], showing how the two similar methods are "equivalent" to some extent. Chapter 4 which presents experiments to illustrate the relationship between setting for SGD and its performance. Next, chapter 5 and 6 explore the two extensions of the vanilla SGD algorithm: the step size adaptation of SGD and the simultaneous multi-quantile estimation. Finally, the work of the paper is summed up in the conclusion chapter 7.

Fig 3 is the layout of the contents of the paper.

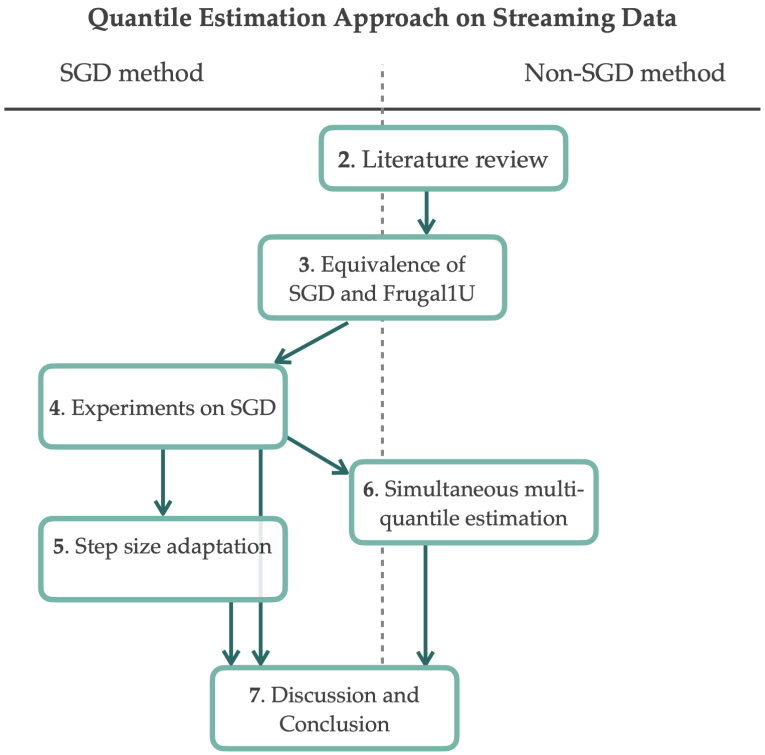

Figure 3: The relationship between topics covered in the thesis. Topics are roughly positioned along the top-bottom axis depending on where they are more close to SGD methods (left) or non-SGD methods (right). The arrows between the chapters represent are connected according to dependence.

2 LITERATURE REVIEW

The field of quantile estimation on data streams is well studied, and this thesis is incapable of covering all the topics. This chapter details some of the work that has been done in this field, and is organised as follows:

Section 2.1 introduces the data stream phenomenon and some quantile estimation methods on it, classified by algorithm type. We specially highlight the space complexity of those algorithms, to show the improvements of quantile estimation algorithms in an intuitive way.

Section 2.2 focuses on algorithms with extreme restriction: estimate a quantile in constant space complexity. Two methods are described in detail, followed by a brief comparison.

Since algorithms in section 2.2 cannot deal with multiple quantiles, section 2.3 introduces an improved constant space usage algorithm for quantile summary queries, which is the inspiration of our work.

2.1 QUANTILE ESTIMATION METHODS ON STREAMING DATA

A data stream is the phenomenon that input data comes at a high rate instead of being accessible all at the same time[26]. Generally speaking, the continuous data can come without a start or end, or consistent time intervals. Streaming data has become popular in industrial application, especially big data use cases, for it "includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers."[34].

Compared with instantly accessible data, three challenges stand out in relation to data streams: transmission, computation and storage[26]. Transmission is difficult when there are frequent delays or the data is in a complicated format, while challenges in computation stem from the need for algorithms to keep up with high rate data streams. Finally, it is infeasible to store all the data produced by large data streams. In this research, we focus only on the latter two problems. We limit our scope to programs with low computational complexity and restricted memory use, under the assumptions that the input data is low-dimensional and is not transmitted at a high rate.

We assume that the data points produced by a data stream are independent identically distributed samples from an unknown distribution. Although access to the distribution is impossible, analysis of a random sample can reveal some useful information. A quantile is an important feature of a distribution, which represents the cutting points which divides a data distribution by its ranking. The computation of a quantile is relatively easy as long as the dataset is sorted. A data stream, however, is updated at the arrival of each single input, hence quantile calculation demands that the data is sorted again every update. What's worse, the unknown number of data points can cause a severe memory short-

age problem for any system. The conclusion now becomes clear that quantile calculation fails to be a feasible solution for data streams.

Quantile estimation replaces calculation for data streams due to the computation and storage problems. The estimation works with limited computational power and memory storage. The batch algorithm for quantile computation takes $O(N)$ space. Generally, quantile estimation algorithms trade off accuracy for computation and storage. Munro and Paterson [25] have shown that any algorithm that computes the exact τ-quantile from an N-element data stream with p passes requires at least $\Omega(N^{\frac{1}{p}})$ space. Therefore quantile estimation algorithms all necessarily aim for sub-linear space. In this chapter, we investigate the accuracy and the memory usage of a variety of existing algorithms for quantile estimation. Section 2.1.1, 2.1.2 and 2.1.3 divide the algorithms by their types: deterministic, randomised and others respectively. The classification is based on works from Buragohain and Subhash[6] and Wang et al.[33].

It is worth noting that since quantiles are related with ranking, a quantile estimation algorithm takes one-dimensional numerical data input by default. The simple data format also partially explains why transmission is not usually considered a problem in most quantile estimation algorithms.

2.1.1 *Deterministic algorithms*

The first deterministic algorithm on quantile estimation was proposed by Manku, Rajagopalan, and Lindsay[22] (referred to as the MRL algorithm), based on the work by Munro and Paterson[25]. Along with the algorithm, the notation ϵ-*approximate* τ-*quantile* is first introduced for accuracy measurement by Manku, Rajagopalan, and Lindsay, which describes the property of guaranteed accuracy on τ-quantile within a pre-specified precision ϵ. The algorithm has a space complexity $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ to compute an ϵ-*approximate quantile summary*. Building on this, two improved algorithms have been proposed by Greenwald and Khanna[13] and Shrivastava et al.[30].

Greenwald and Khanna[13] propose an algorithm (referred to as the GK algorithm) that has a worst-case space requirement of $O(\frac{1}{\epsilon}\log(\epsilon N))$, surpassing the MRL algorithm both theoretically and empirically. The general idea of GK algorithm maintains a sorted subset of data input by the combine and prune operations which keep the precision in control. The space complexity was then improved to $O(\frac{1}{\epsilon}\log^{\frac{3}{2}}\frac{1}{\epsilon})$ by Agarwal et al.[1], who provide a mergeable algorithm based on the GK algorithm.

A deterministic, fixed-universe algorithm *Q-Digest* was designed by Shrivastava et al.[30]. It deals with the quantile problem as a histogram problem using $O(\frac{1}{\epsilon}\log U)$ space, where U is the size of the fixed universe from which the input is drawn.

2.1.2 *Randomised algorithms*

Estimating quantile from a random sample of data stream input is the basic idea for most randomized algorithms, and it reduces space usage to a considerable extent. A simple randomized algorithm designed by Floyd and Rivest[10] uses $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon\delta})$ space, for an ϵ-approximate τ-quantile of N elements with probability at least $1-\delta$, where $0 < \delta < 1$. Later, Manku, Rajagopalan, and Lindsay[23]

proposed a randomized quantile estimation algorithm based on their previous framework in [22]. The new method requires $O(\frac{1}{\epsilon}\log^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon\delta}))$ space, being the first practical algorithm that does not require advance knowledge of data stream size N.

Turnstile models accepts both addition and deletion of data points from a data stream. The turnstile model was first introduced by Gilbert et al.[11], who designed the *random subset sum* quantile estimation algorithm with space complexity $O(\frac{1}{\epsilon^2}\log^2 U\log(\frac{\log U}{\epsilon}))$, where U is the size of the universe. The key idea is based on their findings that insertions or deletions of quantiles can be reduced to finding range sums. The space requirement is greatly improved by another algorithm for turnstile model – the *Count-Min sketch* designed by Cormode and Muthukrishnan[7]. Only $O(\frac{1}{\epsilon}\log^2 N\log(\frac{\log N}{\tau\delta}))$ space is needed for an ϵ-approximate τ-quantile. Although the space requirement is worse than the deterministic algorithms *GK* and *Q-Digest*, it has a noticeable advantage on dealing with both insertions and deletions of data to streams.

Another significant improvement on randomised algorithms were made in 2017, when Felber and Ostrovsky[9] developed an algorithm that uses $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ space. The algorithm not only matches the previous best upper bound of $O(\frac{1}{\epsilon}\log^{\frac{3}{2}}\frac{1}{\epsilon})$ by Agarwal et al.[1], but it also beats the lower bound for any deterministic comparison-based algorithm, according to the work of Hung and Ting[16]. The estimation ensures the return of the ϵ-approximate quantile summary, with probability at least $1-e^{-\mathrm{poly}(1/\epsilon)}$.

2.1.3 *Algorithms for other requirements*

Due to the large range of data streams and quantile application requirements, it is not surprising to have a considerable variety in quantile estimation algorithms attacking specific requirements. We will introduce some interesting applications in the following, such as the methods for biased accuracy requirements and a parallel algorithm for distributed data streams.

Although most quantile estimation work targets the *uniform accuracy* problem on quantile estimation, many queries over data streams require different accuracy for different quantiles. For example, the 0.99-quantile is usually sparsely surrounded while the 0.5-quantile is likely to locate in the center of a distribution where the data is much more dense. Thus comes the *biased quantiles* problem which demands higher accuracy for more extreme quantile values. The first deterministic algorithms for the biased quantiles problem was proposed by Cormode et al. [8] and takes only $O(\frac{1}{\epsilon}\log U\cdot\log\epsilon N)$ space, where N is the size of the data stream and U is the size of universe from which the samples are drawn. Besides, the algorithms also apply to other related problems like biased rank queries and targeted quantiles.

There are also applications that require algorithms to work for distributed systems where a big data stream is processed by different processors. Ben-Haim and Tom-Tov[4] introduce an on-line histogram building algorithm *Streaming Parallel Decision Tree (SPDT)* in which the histogram boundaries can be seen as estimated quantile values. In this method, multiple histograms are built from streaming data in parallel, which are then merged into a summary histogram of the entire dataset. The summary histogram is a set of sorted real numbers

that represents the interval boundaries such that all the intervals contain approximately the same amount of data. Specifically, for a summary histogram with K intervals, the set of real numbers is approximately the set of τ-quantiles ($\tau = \frac{1}{K}, \frac{2}{K}, ..., \frac{K-1}{K}$) for the input data stream. Some quantiles estimation methods mentioned before are also available for distributed settings. To mention a few, Q-Digest[30] is a tree based algorithm which can be extended for distributed quantile processors by merging the quantile summaries across different nodes. The Count-Min sketch[7] is also suited for distributed applications too. In both cases, parallelisation does not change the space complexity. The GK algorithm also has a structure that maintains distributed nodes, and Greenwald and Khanna[12] found a method to fit it in distributed settings with a larger space complexity $O(\frac{1}{\epsilon}\log^3 N)$. It was Agarwal et al.[1] who made the significant improvement to lower the space complexity to $O(\frac{1}{\epsilon}\log^{\frac{3}{2}}\frac{1}{\epsilon})$ with the capability to work in a distributed setting.

In the situation when old data expires, it is desirable to have quantiles of only the most recent observations from a data stream. The *sliding window* represents the most recent W observations in a data stream. The types of sliding window can be classified as either *fixed-size* or *variable-size*. In [20], Lin et al. posed their algorithm for the fixed-size sliding window problem with an $O(\frac{1}{\epsilon}\log(\epsilon^2 W) + \frac{1}{\epsilon^2})$ space requirement. The result was then beaten by the work of Arasu and Manku[2] with space complexity $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log W)$, which are designed for both fixed and variable sized windows. Despite of a worse memory space usage, the work of Lin et al.[20] has the advantage of estimating quantiles for the most w observations ($w < W$) with $O(\frac{1}{\epsilon^2}\log(\epsilon W))$ space requirement.

2.2 NON-GROWING SPACE FOR SINGLE QUANTILE ESTIMATION

The brief history above has shown a clear path for quantile estimation algorithm development: keeping a relatively accurate result while using as little memory as possible. This triggers another interesting question: what if only a constant memory is allowed, how would it affect the accuracy on quantile estimation? In the following part, we introduce two similar algorithms that estimate a single quantile with a $O(1)$ space requirement.

Ma, Muthukrishnan, and Sandler[21] introduce the randomized algorithms frugal streaming (referred to as Frugal) that require only one or two units of memory. This algorithm does not need prior knowledge of the size of input data or the universe, and the query of quantile estimation can be made after the arrival of any item. It is worth noticing that Frugal is significantly different from the randomized algorithms introduced before. As mentioned before, the key idea of randomized algorithms is the maintenance of a random set of samplings from the data stream. Frugal, however, is not able to keep such a samplings with its extremely limited space, its randomness is caused by another design of the algorithm.

The *Frugal-1U* algorithm stores only the current quantile estimate, and the estimate is updated with randomness on the arrival of each new observation. As the authors describe, the current estimate "drifts" in the direction indicated by each new item[21]. That is, when the current estimate equals the new observation, the value is not changed. When the new observation is greater than current estimate,

then the estimate is increased with some probability; otherwise it decreased with some probability. The pseudo code of the algorithm is shown below.

Algorithm 1 Frugal-1U

Input: Data Stream S, τ, 1 unit of memory $\tilde{q}$
Output: $\tilde{q}$

```
1: Initialization q̃ = 0
2: for each s_i in S do
3:     rand = random(0,1); //get a random value in [0, 1]
4:     if s_i > q̃ and rand > 1 − τ then
5:         q̃ = q̃ + 1;
6:     else if s_i < q̃ and rand > τ
7:         q̃ = q̃ − 1;
8:     end if
9: end for
```

Specifically, the output $\tilde{q}$ is the estimate of the τ-quantile for a given data stream S. In line 3~8 of the pseudo code, the algorithm get randomness in its estimate update procedure, for the update is done by probability compared with $1-\tau$ or τ.

To attack the same problem, Yazidi and Hammer[35] resort to the idea of online learning and achieves a similar method. One distinction is step size. In this algorithm, each step the new estimate is updated by a variable related with the current estimate rather than a constant. Their work is inspired by Tierney[31], who introduce stochastic learning method to quantile estimation problems.

2.3 NON-GROWING SPACE FOR MULTI-QUANTILE ESTIMATION

One new problem for the constant space usage algorithms is the limit in number of quantiles estimated for each query. When more quantile numbers are required, for example, two quantiles each identifying the upper and lower outlier of a distribution, the single quantile estimation methods become inefficient and less applicable. For only a few quantiles, this problem can be solved by running parallel single quantile estimation processes for each quantile. But the issue remains when the number of processes exceeds the computer's parallel capacity. This suggests the need for an algorithm which estimates several quantile values in one process, as a general solution to this problem. Multiple quantile estimation, or quantile summary estimation, is the simultaneous estimation of different quantile values from streaming data. In general, it estimates K quantiles (the τ_1-, τ_2-, ..., τ_K-quantiles) at the same time. This is a problem targeted by several different algorithms.

P^2 is another multi-quantile estimation method proposed by Jain and Chlamtac[17]. It assumes a pairwise-parabolic relationship between every three adjacent quantiles, and update quantile estimates based on their relative positions with the neighbour quantiles. Later, an extension version *Extended* P^2 was developed by Raatikainen[27], which changed the initialization of the algorithm to improve the accuracy.

2.3.1 *Other methods and advantages of simultaneous estimation*

Another single-pass low-memory method for simultaneous multi-quantile estimation is the Data Skeleton(DS)[24] algorithm, which is derived from the method proposed by Liechty, Lin, and McDermott [19]. For an estimation of k quantiles, the algorithm requires the first km data points(m is a constant) being sorted, and updates this tracking array on each new observation. Instead of the k estimates of quantiles, it returns a total of km estimates due to the redundancy of computation. This feature is considered an advantage for certain applications like density estimation, when extra quantile estimates is useful in accuracy improvement.

Over the comparison with Liechty, Lin, and McDermott's algorithm, McDermott et al.[24] find simultaneous estimation on multiple quantiles has two main advantages over single quantile estimation methods. First is the reduction in computation time as it does not need to estimate quantiles separately. The second advantage, according to the experiments, is the accuracy improvement in simultaneous quantile estimation.

3 EQUIVALENCE OF SGD AND FRUGAL-1U

In 1978, the *pinball loss* function was proposed by Koenker and Bassett[18], which is well-known for quantile estimation in both statistics and machine learning. In this chapter, we present the implementation of SGD the approach with the pinball loss function for quantile estimation. We also illustrate that, to some degree, there is an equivalence relationship between our algorithm and the state-of-the-art Frugal-1U[21] by both theoretical analysis and experimental results. After the introduction of pinball loss in section 3.1, section 3.2 introduces the SGD methods and at the same time shows the equivalence to Frugal-1U by rephrasing the algorithm. In addition, we offer some complementary explanation, supporting experimental results and further discussion for the proof of algorithm equivalence in section 3.3.

3.1 PINBALL LOSS FOR QUANTILE ESTIMATION

The pinball function is a convex loss function for estimation of a quantile value. For a one-dimensional dataset $X = \{x_1, x_2, \cdots, x_N\}$, consider the loss function for a single data point x_i $(i \in 1, \cdots, N)$ of the τ-quantile ($\tau \in (0, 1)$). Let $t := x - q$ be the difference between the real value x and the estimate of quantile q. The pinball loss function $\ell_\tau(\cdot) : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ on t is defined by

$$\ell_\tau(t) = \begin{cases} \tau t & t > 0 \\ -(1-\tau)t & t < 0 \end{cases} \tag{12}$$

And the τ-quantile loss has the subgradient:

$$\frac{\partial \ell_\tau(t)}{\partial t} = \begin{cases} \tau & t > 0 \\ [\tau, -(1-\tau)] & t = 0 \\ -(1-\tau) & t < 0 \end{cases} \tag{13}$$

Note here we use subgradient instead of gradient, because the pinball loss function is not differentiable at 0. In the practical implementation of SGD, we choose the the subgradient at 0 to be 0. Since the possibility of reaching the 0 point is 0, the choice of replacement would not influence the final results. Empirically it is acceptable in this thesis that we ignore the breaking point at 0 and take this subgradient function as the gradient of pinball loss function.

The overall loss for distribution X with quantile estimation q is

$$L_\tau(q) = \sum_{x \in X} \ell_\tau(x - q) \tag{14}$$

The best estimate of the τ-quantile q is the q with minimal overall loss. Let q^* be the best estimate, then we have

$$q^* = \arg\min_q L_\tau(q) \tag{15}$$

3.2 DERIVING THE SGD APPROACH FROM FRUGAL

Inspired by Frugal-1U[21], we propose a quantile estimation method using the SGD approach with implementation of the pinball loss. Here this algorithm is introduced by rephrasing Frugal-1U, which also provides an intuition for the equivalence of the two algorithms.

3.2.1 *Pseudo Code for the Frugal-1U Algorithm*

As a quantile estimation algorithm using constant memory storage, Frugal-1U reaches the extreme minimum of memory usage, as it "uses only one unit of memory per group to compute a quantile for each group"[21]. Besides the minimal memory usage, the update method is also computationally simple. The pseudo-code of Frugal-1U is presented as algorithm 2.

Algorithm 2 Frugal-1U

Input: Data Stream X, k, Q, 1 unit of memory q
Output: q

1: Initialization $q = 0$
2: **for each** x_i in X **do**
3: $rand$ = random(0,1) ▷ get a random value in $[0, 1]$
4: **if** $x_i > q$ **and** $rand > 1 - \frac{k}{Q}$ **then**
5: $q = q + 1$;
6: **else if** $x_i < q$ **and** $rand > \frac{k}{Q}$
7: $q = q - 1$;
8: **end if**
9: **end for**

The output q is the estimate of the kth Q-quantile (see section 1.4.2) for a given data stream X. By rephrasing of some steps of Frugal-1U, its equivalence to an SGD algorithm for quantile estimation will be shown in the following part.

Rephrasing of the Algorithm

1. The constant $\frac{h}{k}$ is replaced by τ, since the τ-quantile is defined as the hth k-quantile point in chapter 1.

2. The generation of the random number and it's comparison with $1 - \frac{h}{k}$ or $\frac{h}{k}$ in line 3 to 7 is replaced by the following algorithm.

3: ▷ No need to generate a random number
4: **if** $x_i > q$ **then**
5: $q = q + 1 \times (1 - \frac{h}{k})$ ▷ $P((rand > 1 - \frac{h}{k}) \mid rand \in \mathcal{U}(0, 1)) = 1 - \frac{h}{k}$;
6: **else if** $x_i < q$
7: $q = q - 1 \times \frac{h}{k}$ ▷ $P((rand > \frac{h}{k}) \mid rand \in \mathcal{U}(0, 1)) = \frac{h}{k}$;

To understand this replacement, consider the 3 steps:

(i) generate a random number $rand$,

(ii) compare it with a constant p, and

(iii) take action if $rand > p$.

This can be interpreted as "take the action with probability $P((rand > p) \mid rand \in \mathcal{U}(0,1))$".

While the behaviour is different, this replacement preserves the expected change of q. For example when $x_i > q$, the expected change of q is $E_1[\nabla q] = E[q \times p]$ in the Frugal-1U with random number generation, while $E_2[\nabla q] = q \times p$ in the replacement method. Since $E_1[\nabla q] = E_2[\nabla q]$, the replacement is valid with regard to the expectation of the change in quantile estimate during each step.

3.2.2 *SGD for pinball loss function*

Let q_0 be the initial guess of the quantile estimate. By SGD, the estimate is updated each step with a data point from the distribution.

$$q_{k+1} = q_k - \alpha_k g_k \tag{16}$$

where α_k is a suitable step size and

$$g_k = \partial L_\tau^{(k)}(q_k) \in \frac{\partial l_\tau(x_k - q_k)}{\partial q_k} \tag{17}$$

Notice here partial is taken because the gradient of a single variable function equals the partial of it.

Then we have

$$q_{k+1} = \begin{cases} q_k + \alpha_k \tau & x_k - q_k > 0 \\ q_k & x_k - q_k = 0 \\ q_k - \alpha_k (1-\tau) & x_k - q_k < 0 \end{cases}$$

Algorithm 3 SGD algorithm

Input: Data Stream X, τ, 1 unit of memory q
Output: q

```
Initialize q                                   ▷ Default initialization q_0 = 0
for x_k in X do                                ▷ Parameter update for each input data point
    set α_k = 1                                ▷ Set step size
    if x_k > q then                            ▷ q_{k+1} = q_k + α_k τ when x_k − q_k > 0
        q = q + α_k τ
    else                                       ▷ q_{k+1} = q_k − α_k (1 − τ) otherwise
        q = q − α_k (1 − τ)
return q                                       ▷ q_k is the SGD result of quantile estimate
```

Besides the replacements mentioned in section 3.2.1, the notation has changed from Frugal-1U to SGD: X is used instead of S for the data stream. The other important change is the introduction of step size. Specifically, the step size α is not mentioned in Frugal-1U because it is fixed as constant 1. On the other hand, the choice of constant or variable step size α_i is an important feature of the SGD algorithm. The flexibility of step size can contribute to a better convergence rate when appropriate step sizes are chosen.

3.3 EQUIVALENCE OF ALGORITHMS

The derivation of the SGD algorithm from Frugal-1U has intuitively shown the evident similarity between them. In this section, we investigate the similarity between our approach, which we refer to as the SGD algorithm, and Frugal-1U, which leads to the proof of equivalence by experimental performances with additional discussion.

3.3.1 *Comparison experiments between SGD and Frugal-1U*

We investigate various aspects of the similarity between the SGD algorithm and the Frugal-1U algorithm: behaviours and results on different data distributions. The behavioural similarity is inspired by the *behavioural equivalence* [14], which requires sequential algorithms to have same states, same initial states, and the same transition function when converted to an abstract state machine (ASM). The quantile estimation results are compared based on the final output results of the algorithms.

3.3.1.1 *Data distributions and algorithm settings*

For comprehensiveness, we evaluate the algorithms using data generated from three distributions:

1. a Gaussian distribution with mean 2 and standard derivation 18 (*gau-1*)
2. a mixed Gaussian distribution composed of 5 Gaussian distributions (*mix*):
   a) weight = 0.3, mean = 2, standard derivation = 7
   b) weight = 0.2, mean = 0, standard derivation = 0.7
   c) weight = 0.1, mean = 36, standard derivation = 26
   d) weight = 0.15, mean = 5, standard derivation = 77
   e) weight = 0.25, mean = -77, standard derivation = 7
3. a exponential distribution of rate 1, and is scaled by 6.5 and shifted by -20

The algorithms are initialised with the starting estimate for any quantile value at 0. The step size of the SGD algorithm is set as constant 1 to match the implicit step size of Frugal-1U. Each data stream contains 1000 randomly ordered samples from the corresponding distribution with a fixed sequence. The estimation on the $0.1, 0.3, 0.5, 0.9$, and 0.99 quantiles are checked.

3.3.1.2 *Evaluation methods*

For each simulation setup, we run the Frugal-1U algorithm 100 times and SGD once. Since the data stream is a fixed sequence of data points, the behaviour and result of SGD are also fixed, whereas Frugal-1U is likely to perform differently given its randomness in each update step. For a quantile probability τ, let τ-q_{batch} be the computation results of the batch quantile, τ-q_{SGD} and τ-q_{frugal} be the results respectively of SGD and Frugal-1U algorithm. The similarity of the two estimates S_τ is measured by

$$S_\tau = \frac{\tau\text{-}q_{SGD} - \tau\text{-}q_{frugal}}{\tau\text{-}q_{SGD} - \tau\text{-}q_{batch}} \tag{18}$$

where a smaller $|S_\tau|$ means a higher similarity, and $|S_\tau| = 0$ means the two quantile estimates are the same. In this paper, we consider $\leqslant 2^6 = 64$ as the threshold for the similarity error, that is, if $|S_\tau| \leqslant 2^6$, we believe the the two results are similar enough to be viewed as "equal".

Both algorithms start the quantile estimate at 0. When a new observation arrives, Frugal-1U updates the value by 1 or 0 according to probability, while the update is a deterministic floating number for SGD. The behavioural similarity is compared at each update step. For Frugal-1U, we take the mean value of the quantile estimate update over 100 runs, and it is compared with the one fixed SGD value at the same epoch.

3.3.1.3 *Results for similarity comparisons*

The mechanism for data generation is as follows. First, generate a dataset with 1000 samples according to one of the three distributions specified in section 3.3.1.1. For each distribution, we also compute the true quantiles. The sequence of the dataset is fixed for all the experiments, including one computation for the batch quantiles, one for the SGD quantiles, and 100 for the Frugal-1U quantiles. The process of quantile estimation is shown in the process plots (fig 4, 6, and 8), in which we can see the comparison in algorithm behaviours. The true quantiles are also shown in these plots, as a reference to the algorithm performances. Figure 5, 7, and 9 show the similarity error values of the final results only.

It is also worthwhile to point out that while 1000 is not a large size for data streams, it is sufficient to evaluate the similarity between the two algorithms as it does not need to show the convergence to the true quantile. In fact, the convergence of most quantile probabilities over the three datasets are already reached by the 1000 epochs.

The process plot in fig 4 comparing Frugal-1U and SGD shows that the mean of Frugal-1U steps and the SGD functions are largely overlapped. Take the 0.1-quantile for example, the two algorithms start at the same point, and the mean Frugal-1U line shares similar trends and close quantile estimate position with SGD. They even overlap in many intervals (where the dark blue line is covered by the light blue line). Irrespective of the data distribution and the quantile probabilities, it is clearly shown in figure 4, 6, and 8 that the quantile estimation behaviours of mean Frugal-1U and SGD are very similar. However, this evidence is not sufficient to claim a close similarity because the randomness of Frugal-1U might lead to a large variety in behaviours.

For a closer examination, the similarity error values of the quantiles are checked at the result states. Figure 5 illustrates the distributions of similarity error values between the 100 runs of Frugal-1U and the one SGD result on *gau-1*. Take the 0.1-quantile for example, the similarity error values are distributed in the range $(-2^4, 2^3)$, which means $|S_\tau| < 2^6$ for all 100 runs. Furthermore, the mean of the similarity error (the vertical black line) has an absolute value less than 1, indicating the mean Frugal-1U result and the SGD result is nearly the same. The results from the three figures (fig 5, 7, and 9) also imply the high similarity of Frugal-1U and SGD works for all three distributions and all five quantiles checked in the experiment.

Results from the experiments (fig 4 to 9) support our theoretical conjecture that the two algorithms are highly similar in both behaviours and results regardless of distributions or quantile probability values. The similarity is high enough to

meet our requirement for the equivalence (absolute similarity error value less than or equal to 64).

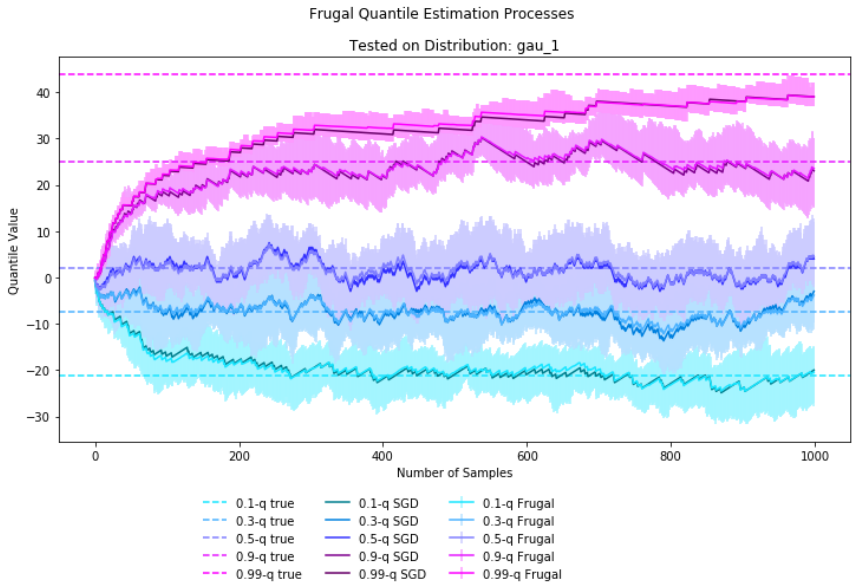

Figure 4: Quantile estimation behaviour similarity between Frugal-1U and SGD on the *gau-1* dataset. The horizontal dotted lines are the true quantiles, the dark solid colour lines are SGD quantiles, and the lighter solid colour lines are the mean value of Frugal-1U quantiles. The light coloured shadows around the solid lines are the range of Frugal-1U steps over the 100 runs. For each epoch, the top and the bottom margin of the shadow are positioned respectively on the maximum and minimum value of the 100 experiments. The shadow of a quantile is an indicator of how much the Frugal-1U estimate varies around the mean value. Unfortunately the shadows from different quantiles sometimes cover each other, so the it can only be used as a rough visual reference of changes in Frugal-1U.

3.3.2 *Discussion about the equivalence*

The comparison experiments between Frugal-1U and SGD implies there is a high similarity relation between them with regards to both behaviours and results. Although it is claimed in this paper that the SGD algorithm and the Frugal-1U algorithm are equivalent to some extent, Blass, Dershowitz, and Gurevich[5] have already pointed out that there is no such relation as equivalence between any two algorithms.

The claim that the two algorithms are equivalent is based on the check of similarity error value. It is obvious that if the requirement for similarity error value is more strict, e.g. $|S_\tau| \leqslant 2^2$, then the equivalence would no longer hold for some distribution and quantile value settings. That is what we refer to as "equivalence to some extent".

3.4 CONCLUSION

The SGD algorithm is proposed by derivation from the Frugal-1U algorithm. We performed an empirical comparison between Frugal-1U and SGD over 3 different

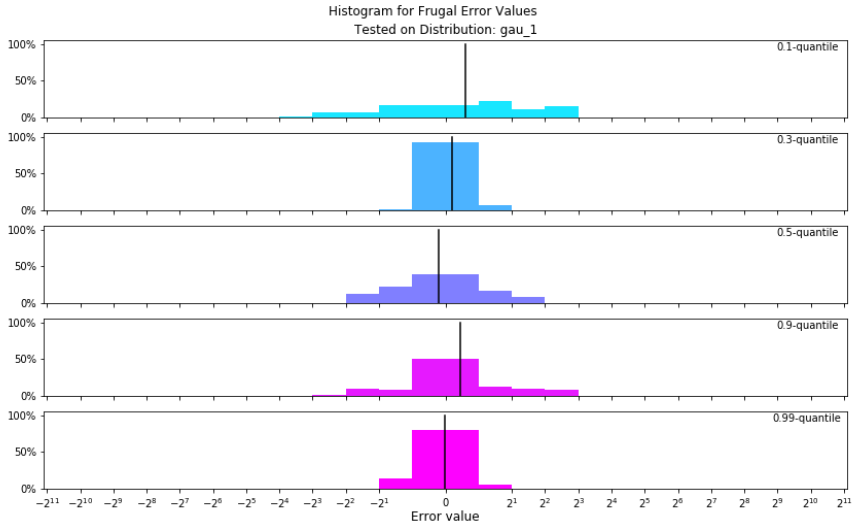

Figure 5: Quantile estimation result similarity between Frugal-1U and SGD on the *gau-1* dataset. Each of the 5 histograms represents the distribution of Frugal-1U similarity error values for a specific quantile. The similarity error values are compared by function (18). For example, consider the top one (the 0.1-quantile). The distribution of 100 similarity error values are shown in the bright blue histogram. The mean similarity error value is also in the form of the vertical black line.

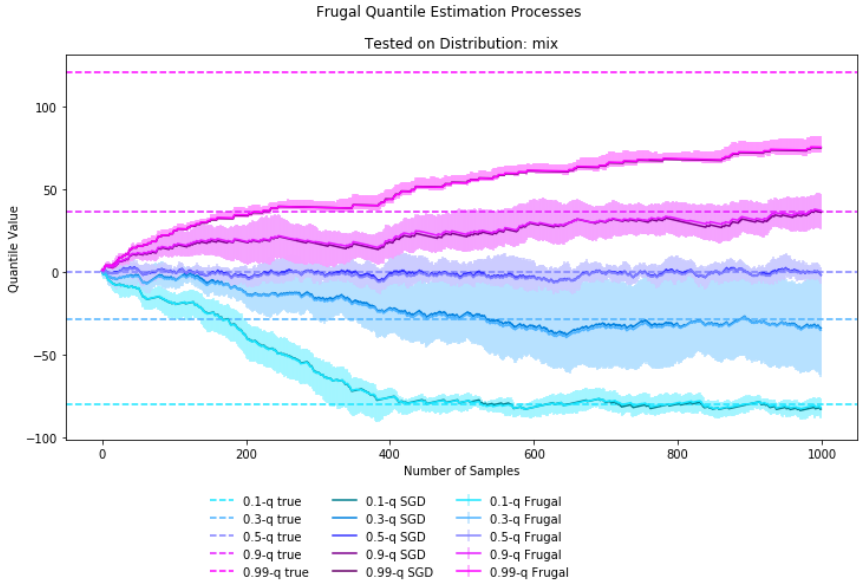

Figure 6: Quantile estimation behaviour similarity between Frugal-1U and SGD on the *mix* dataset

distributions. In addition, we had a theoretical discussion about the equivalence between them. The two algorithms are shown to be equal to some extent. As Frugal-1U is a valid quantile estimation method, we deduce that SGD is a valid quantile estimation method by equivalence. The main difference is that SGD is overall more stable than Frugal-1U. Although SGD is not random for a fixed sequence data stream, it benefits in flexibility in the form of an adjustable step size. In the next chapter we investigate various settings of SGD in more detail.

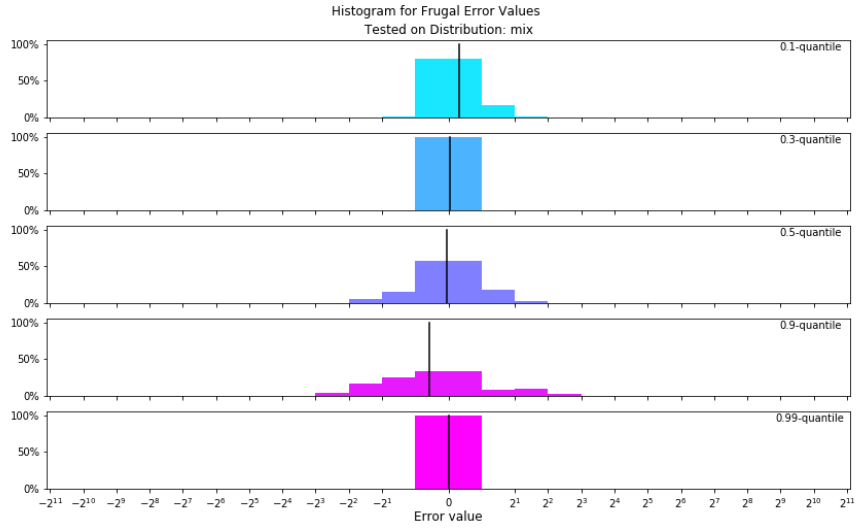

Figure 7: Quantile estimation result similarity between Frugal-1U and SGD on the *mix* dataset

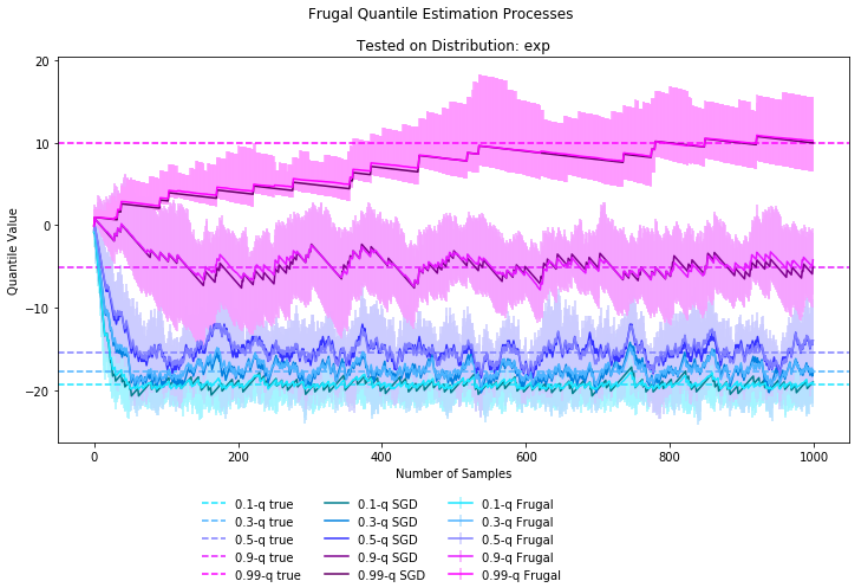

Figure 8: Quantile estimation behaviour similarity between Frugal-1U and SGD on the *exp* dataset

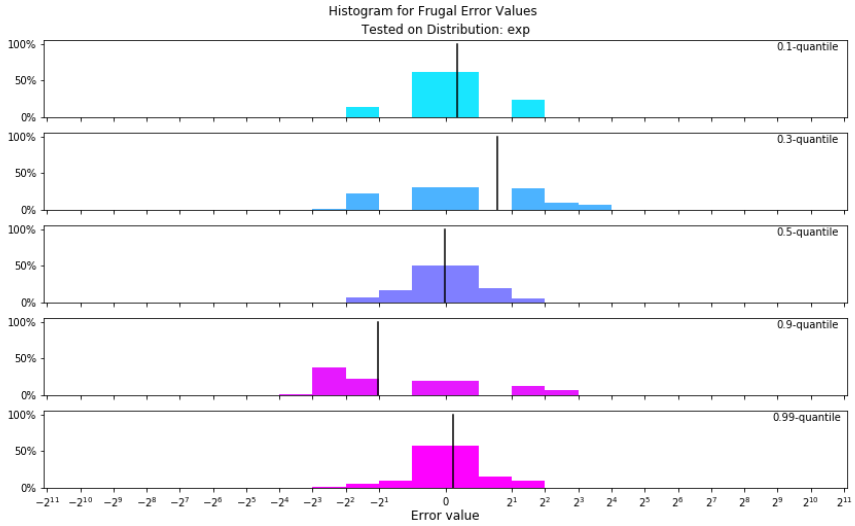

Figure 9: Quantile estimation result similarity between Frugal-1U and SGD on the *exp* dataset

4

EMPIRICAL INVESTIGATION OF SGD CONVERGENCE

As the previous chapter has shown the equivalence between Frugal-1U and SGD, which means the SGD is a valid alternative for quantile estimation. In this chapter, we further analyse the SGD algorithm with more experiments. The experiments have two general purposes. The first is to explore whether SGD quantile estimation works. The second is to investigate how different settings of the problem effect the estimation performance. Specifically, we are interested in the following aspects: data distribution, data size, data ordering, quantile property value and step size. The content of the chapter is listed as follows:

Section 4.1 introduces the basic settings of the experiments, including the generation of various input data streams and SGD with different step size settings.

Section 4.2 explains the evaluation methods for the experiments, with a detailed reasoning of error evaluation and a brief mentioning of plots.

Section 4.3 illustrates the experimental results on 4 different aspects with plots, and analyses the causes of variance in performances.

Section 4.4 provides further insights into the influential factors of SGD convergence, which leads to the conclusion in 4.5 on SGD improvements.

4.1 METHODOLOGY

The process by which we experiment on SGD quantile estimation can be briefly outlined as follows:

1. Generate an SGD quantile estimate of benchmarking setting.
2. For each setting category, generate the other quantile estimate with accordingly changed settings.
3. Compare the performances of both groups of estimate value.

For step 2, the detailed generation steps for a quantile estimate are:

(i) Select a set of data streams (ordered datasets) derived from some statistical distributions.

(ii) For each τ-quantile, determine a ground truth value from the distribution and calculate a batch quantile value from the data stream.

(iii) For each τ-quantile, calculate the SGD estimate value from the data stream, record both the process and the result of estimation.

(iv) Compute a normalized error value for quantile estimates as a measurement of similarity between batch and estimate value. The error value is computed from both values.

4.1.1 *Benchmarking Estimate Setting*

The benchmarking setting for the SGD quantile estimate is:

1. Data stream:
    - Distribution: Gaussian distribution (*gau-1*): mean = 2, standard deviation = 18
    - Data size: 1000 independent and individual distributed random samples
    - Multiple generations: true. 10 data streams of the same setting are generated
    - Multiple shuffles: false. No data streams are generated as a shuffled version of another
2. SGD settings:
    a) Step size: constant value 1
    b) Quantile estimate initialization: 0 for all $\tau \in (0, 1)$

We will also refer to this as default settings for SGD.

4.1.2 *SGD Quantile Estimate Generation*

To ensure the consistency of quantile estimate construction, the generation methods for a specific setting are restricted. The experiment generation here is very similar to the equivalence experiments between Frugal-1U and SGD, but some details are different, for example, the introduction of a new distribution. In this section, we will introduce the rules and steps for generation in details. This series of settings are also of importance for the next chapters, as they are repeatedly used for their experiments.

4.1.2.1 *Data Stream Set Generation*

A total of 4 distributions are used in this experiment. Each data stream is a set of 1 dimensional data points randomly sampled from one of the distributions. In order to show how the amount of data points might affect the performance, there are 3 different settings for the data size N.

Each data stream set is composed of a number of data streams. For a statistically more accurate results on the experiment, a group of data streams with the same settings are generated. When investigating the impact of data sequence has on quantile estimation, one data stream will be shuffled to for the generation to differently ordered data steams. To sum up, a data stream set is either a combination of data streams generated from same distribution and data size setting, or the permutations of one same data stream. We generate the data stream set under this settings:

- Distribution: 4 statistical distributions. The 4 distributions are:
    - Gaussian distribution 1 (*gau-1*): mean = 2, standard deviation = 18
    - Gaussian distribution 2 (*gau-2*): mean = 0, standard deviation = 0.001
    - Exponential distribution (*exp*): rate = 1
    - Mixed Gaussian distribution (*mix*): a mix of five different Gaussian distributions,

    * weight = 0.3, mean = 2, standard derivation = 7
    * weight = 0.2, mean = 0, standard derivation = 0.7
    * weight = 0.1, mean = 36, standard derivation = 26
    * weight = 0.15, mean = 5, standard derivation = 77
    * weight = 0.25, mean = -77, standard derivation = 7
- Data size: 100, 1000, 100000
- Multiple generations: True or false. Generate 10 data streams for the set if true.
- Multiple shuffles: True or false. Shuffle the data stream 10 times for the set if true.

4.1.2.2 *True and Batch Quantile Calculation*

The true quantile values are the quantile values for the distributions which the data streams are derived from. The true quantile values can easily be calculated analytically for each distribution except the Gaussian mixture distribution. For the mixture distribution, the batch quantile value from a large number of samples is taken as a reasonable approximation of the true value. By this means, the empirical value is expected to be close enough to the true quantile value such that the evaluation of results is not much affected. In this experiment, a total of 100,000,000 samples are generated for the true quantile calculation. For a certain τ, there is only one true quantile value for one distribution.

The batch quantile value is the quantile value calculated from the data steam instead of the distribution. For a certain τ, no matter what the ordering of the data stream is, there is only one batch quantile value for one data stream, but there can be multiple quantile values for one distribution.

4.1.2.3 *SGD Quantile Estimation*

The parameter of SGD quantile estimation is important.The current settings for step size α_i are:

- Constant number: $\alpha_i = 1$
- Decrease as k increases: $\alpha_i = \frac{2}{\sqrt{i}}$
- Decrease as k increases (smaller size): $\alpha_i = \frac{0.002}{\sqrt{i}}$

where i is the index of step count.

4.2 EVALUATION METHODS

In our experments, we evaluate the effectiveness of SGD on various settings by comparing the quantile estimates to the batch quantiles and the true quantile values. We also use several standard plot formats to display the results of each experiment compactly.

4.2.1 *Error Computation*

An error measurement is proposed in order to evaluate the performance of quantile estimation. The error value represents the difference between the batch quantile and estimated quantile value. For one data stream, the error function for its τ-quantile is first defined as $E^{(\tau)} = |q_{batch}^{(\tau)} - q_{sgd}^{(\tau)}|$, where $E^{(\tau)}$ stands for the error, $q_{batch}^{(\tau)}$ for batch quantile value and $q_{sgd}^{(\tau)}$ for SGD estimate value. For a specific data stream, a smaller $E^{(\tau)}$ means the estimation for the τ-quantile has a better accuracy. Generally, for n data streams of the same size and distribution, we take the mean of the error value:

$$\overline{E^{(\tau)}} = \frac{1}{n}\sum_{i=1}^{n} E_i^{(\tau)} \tag{19}$$

where $E_i^{(\tau)}$ is the error value of τ-quantile for the ith data stream. To compare the performance of different settings of SGD estimation, we can now compare the $\overline{E^{(\tau)}}$ value for each setting.

This method leaves some room for improvement. First, the limitation of data distribution. The comparison is only available for data streams generated from the same distribution, since a different distribution has difference density of data points for the same τ value, leading to a failure of error comparison. For example, a data stream generated from uniform distribution $\mathcal{U}(0,1)$, the error value $\overline{E^{(0.1)}} = 2$ is a bad estimation, because 2 is even greater than the difference between the minimal and maximal value of the distribution ($2 > |0-1|$). However, for a data stream sampled from uniform distribution $\mathcal{U}(0, 10^{10})$, $\overline{E^{(0.1)}} = 2$ might be a really accurate result, given how low the density is around its 0.1-quantile. Second, the variability of error rates between different τ values. Similarly with the distribution problem, different τ values in the same distribution may have varied density. For example, for a Gaussian distribution, $\overline{E^{(0.01)}} = \overline{E^{(0.5)}}$ means that the estimation is better for 0.01-quantile than 0.5-quantile, since the distribution is denser around the middle than its outlier. Third and more importantly, it is incapable of showing if the estimation "works". Specifically, for some number x, we cannot find a reasonable explanation for the statement "the estimate is accurate enough because we have $\overline{E^{(\tau)}} \leqslant x$". Since for any x, we could find an example from the first two issues as a counter example. To solve those problems, we provide another error measurement which gives a more general accuracy comparison.

In the new version of error value calculation, true quantile value of a data distribution $q_{true}^{(\tau)}$ is involved, so that $E^{(\tau)}$ is normalized by $q_{batch}^{(\tau)} - q_{true}^{(\tau)}$. It is now defined as

$$E^{(\tau)} = \frac{q_{batch}^{(\tau)} - q_{sgd}^{(\tau)}}{q_{batch}^{(\tau)} - q_{true}^{(\tau)}} \tag{20}$$

So that the accuracy of $q_{sgd}^{(\tau)}$ is compared with the accuracy of $q_{batch}^{(\tau)}$. The above problem is solved because $q_{batch} - q_{true}$ is the normalization scaler

which reduces the effect of unevenly distributed density among different distributions and different quantile values.

4.2.2 *Performance Comparison*

The performance comparison between different settings in this thesis are shown in three plots, which respectively represent the quantile estimate processes, results and the error values.

- Process plot: shows the processes of the SGD algorithm. (See explanation under figure 4 for more details)
- Result plot: shows the estimate results of q_{batch} and q_{sgd} over multiple runs, provides a clear visualization of the clustering of both values.
- Error plot: shows the normalized error value for every data stream of this setting. (See explanation under figure 5 for more details)

All of them show the summarized visualization of multiple generation of estimate values.

4.3 EXPERIMENT OBSERVATIONS

The investigation of SGD performances focuses on the setting change in the following 4 areas:

1. Data distribution of data streams (subsection 4.3.1)
2. Data size of the data streams (subsection 4.3.2)
3. Data sequence of the the data streams (subsection 4.3.3)
4. Step size of the SGD algorithm (subsection 4.3.4)

In each of the subsections we show the 3 performance plots for every setting, and make analysis based on the the observations and comparisons. The plots for each experiment are arranged in the order of processes, results then error values. This is because the processes show the trend of overall convergence, then the results show the detailed version of final output of the algorithm and in the end the error values provide a quantitative evaluation of performances.

4.3.1 *Distribution*

The experiment compares the SGD algorithm performance in 4 different data distribution settings: *gau-1*, *gau-2*, *mix* and *exp*. In each setting, 10 data streams of size 1000 are generated from the according distribution, and the SGD is run on each of the data streams. The SGD algorithm is set to the default setting. That is, the performance of one distribution setting is evaluated on the collection of SGD implementations on its 10 data streams.

The process, result and error plots for distributions (figure 10 to 21) leads to the following interesting observations:

- The convergence of SGD is illustrated by all the valid experiments (all but *gau-2*). Figure 10, 12 and 13 show the clear trend that the average SGD quantile estimate of each quantile are approaching to the true quantile values. The result plots show the result of SGD estimation, which reflect the convergence in the corresponding process plots. For example, the 0.99-q SGD for the *mix* distribution has not reached the true quantile value, and it is reflected in figure 16 that the 0.99-q SGD estimates are further away from the batch quantiles. Figure 11 is too messy to be used as a illustration on the convergence performance of SGD, for the quantile estimates frequently cross each other. Specifically, the information in figure 11 shows the quantile values are all within the range $(-2, 2)$, but it cannot imply if any SGD quantile is converging.

- Overall, SGD converges at different rates on different distributions. Specifically, the proportion of quantile estimates that have reached their true quantile value is different. For example, we can see from figure 12 that only 2 of 5 SGD quantiles(0.5-q and 0.1-q) have converged, while in figure 10 there are 4 quantile estimates have already converged and the last one (0.99-q) is very close to reach the convergence value.

- The convergence rate for different quantiles in the same distribution are different. Take the *exp* distribution for example, in figure 13, the $0.1, 0.3, 0.5$-q_{SGD} estimates reach their convergence value within the first 100 epochs,

while it takes the 0.9-q_{SGD} around 300 epochs and 0.9-q_{SGD} over 1000 epochs. This happens even though the 0.1-q_{True} (distance $> |-20-0| = 20$) is further away from the initialization point 0 than 0.99-q_{True} (distance $< |10-0| = 10$).

- The effectiveness of SGD is different for different distributions or quantiles. According to the error figure 20 and 19, the error value of SGD estimates are within the range $(-2^7, 2^6)$ for the *mix* distribution, while they span $(-2^{11}, 2^8)$ for the *gau-2* distribution. In the same distribution *gau-1* (figure 19), the error values for 0.99-q SGD are centred at $(-2^1, 2^2)$, while that of the 0.5-q SGD can be more than 2^{11} (which is not shown in the histogram, but can be inferred from the position of the mean error). Although the error value distributions vary a lot in different circumstances, the mean SGD error is almost always limited by $(-2^4, 2^4)$. The exceptions are the 0.5-q SGD of *gau-1* (which is affect by one extreme value) and all of *gau-2* (special case, discussed in the following part).

- For distribution *gau-2*, the default setting SGD is not an ideal quantile estimation approach no matter if it converges or not. Although all the SGD quantiles are stable within range $(-2, 2)$ centering at 0, the true quantiles are in a much smaller range around 0. The big changes of SGD update at each step is not able to adjust for such small differences between different quantiles. To make it worse, it is clear that at the end of the 1000 epochs, the 0.9-q_{SGD} and 0.99-q_{SGD} still frequently go across each other. Further discussion of the behaviour of SGD on *gau-2* is provided in section 4.4.

4.3.1.1 *Estimate processes on different distributions*

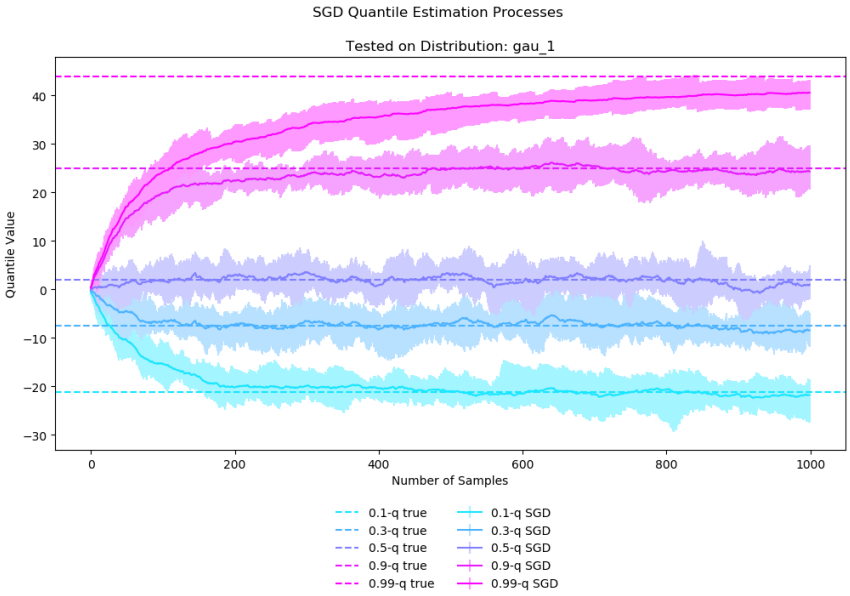

Figure 10: SGD Process from *gau-1* Distribution

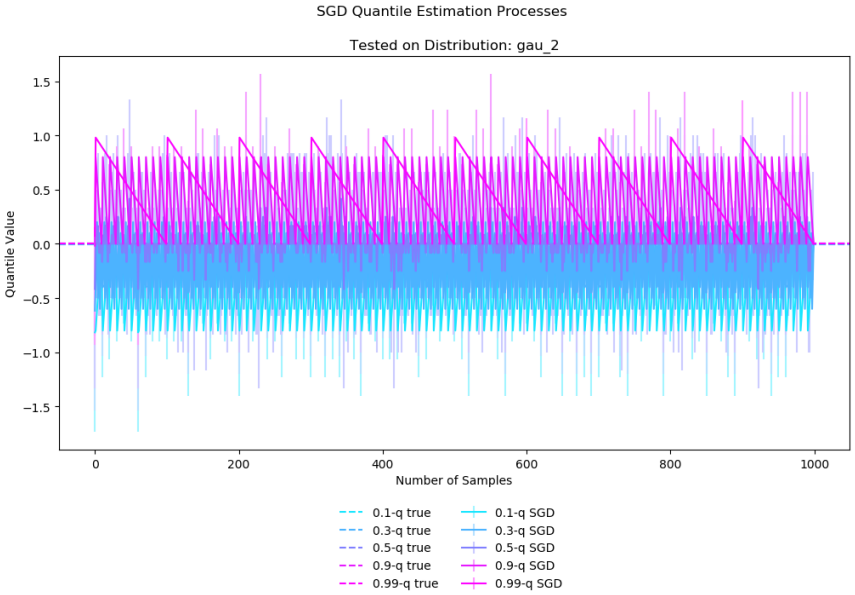

Figure 11: SGD Process from *gau-2* Distribution

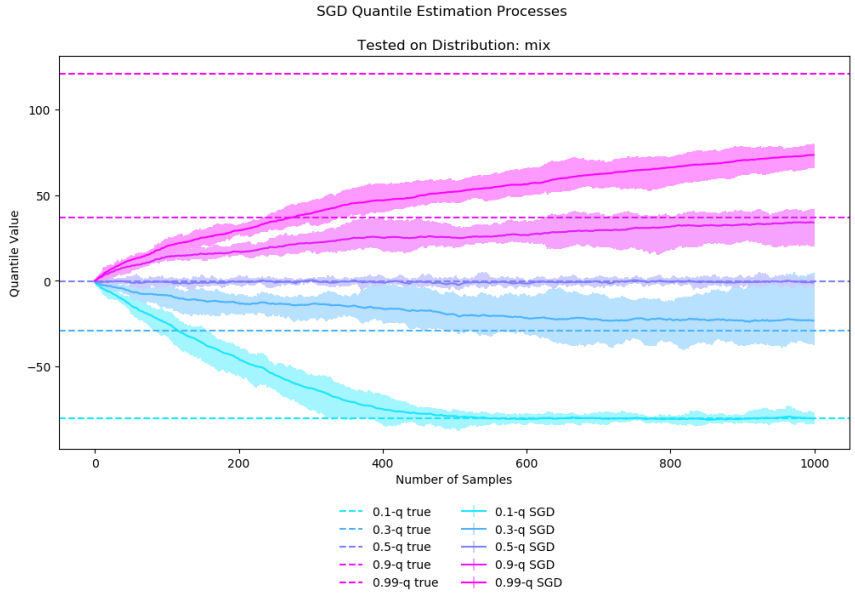

Figure 12: SGD Process from *mix* Distribution

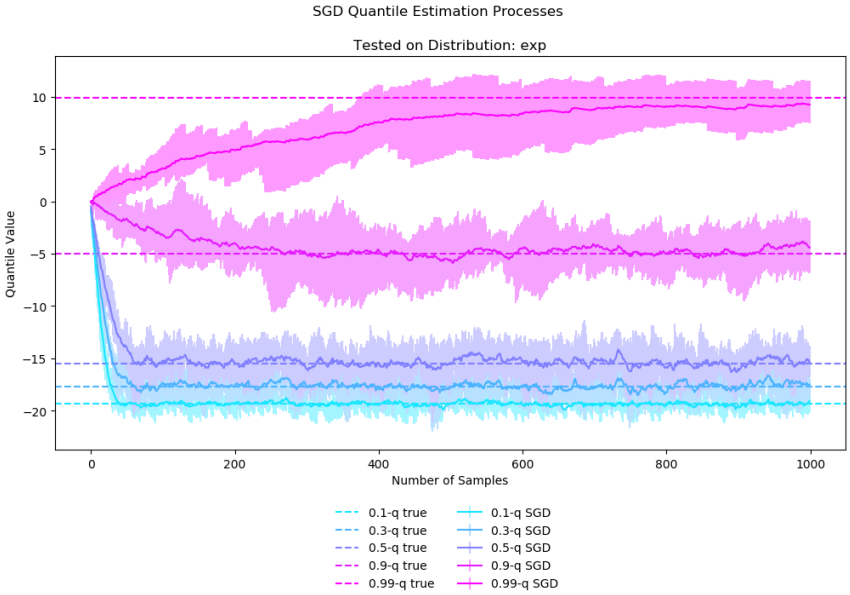

Figure 13: SGD Process from *exp* Distribution

4.3.1.2 *Estimate results on different distributions*

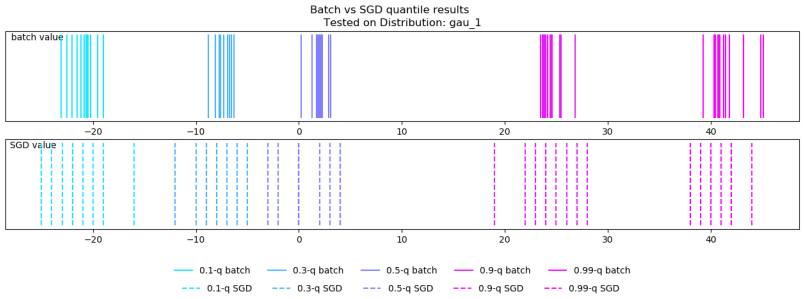

Figure 14: SGD Results from *gau-1* Distribution

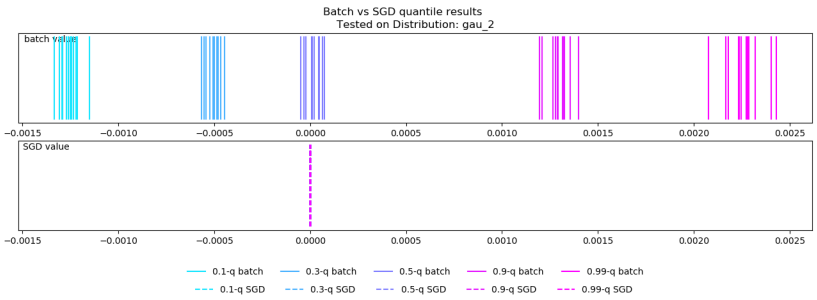

Figure 15: SGD Results from *gau-2* Distribution

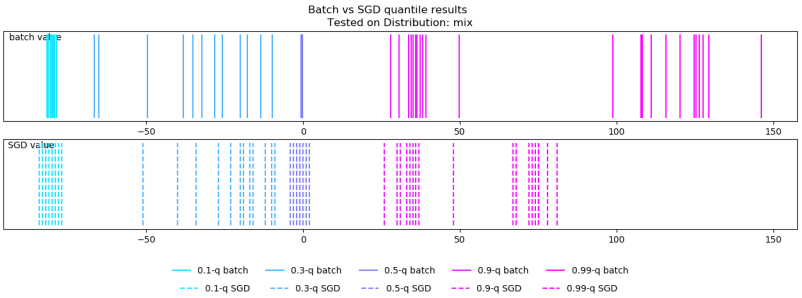

Figure 16: SGD Results from *mix* Distribution

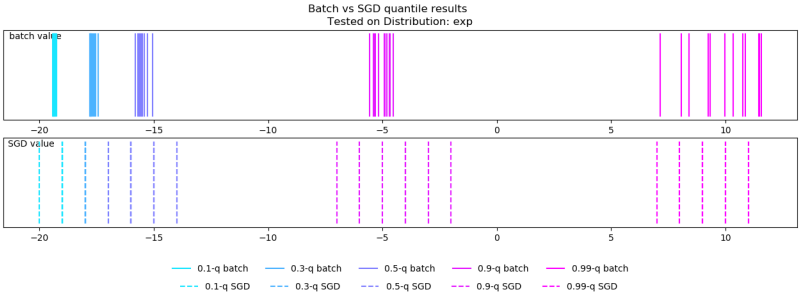

Figure 17: SGD Results from *exp* Distribution

4.3.1.3 *Estimate Error Value on different distributions*

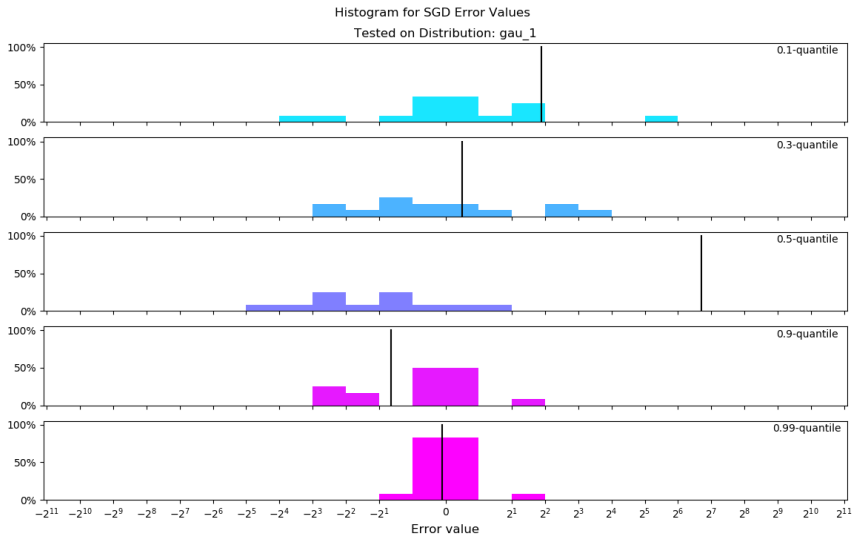

Figure 18: SGD Error from *gau-1* Distribution

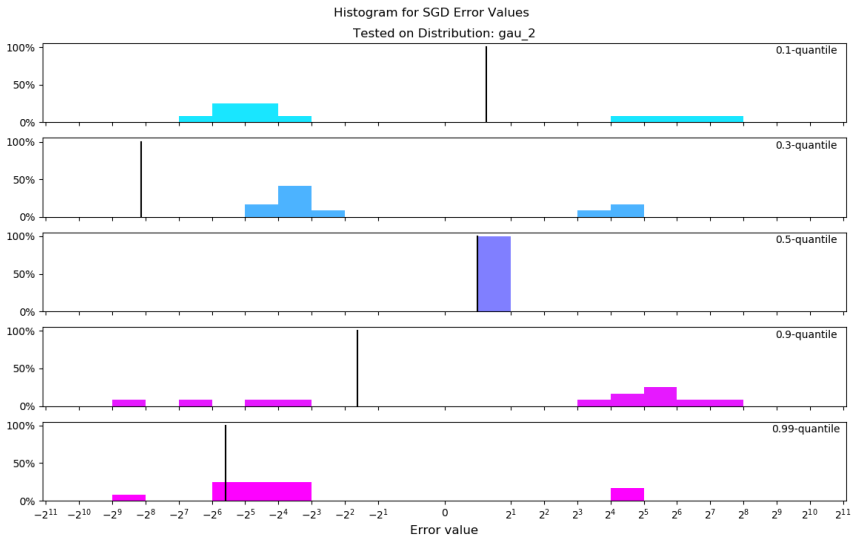

Figure 19: SGD Error from *gau-2* Distribution

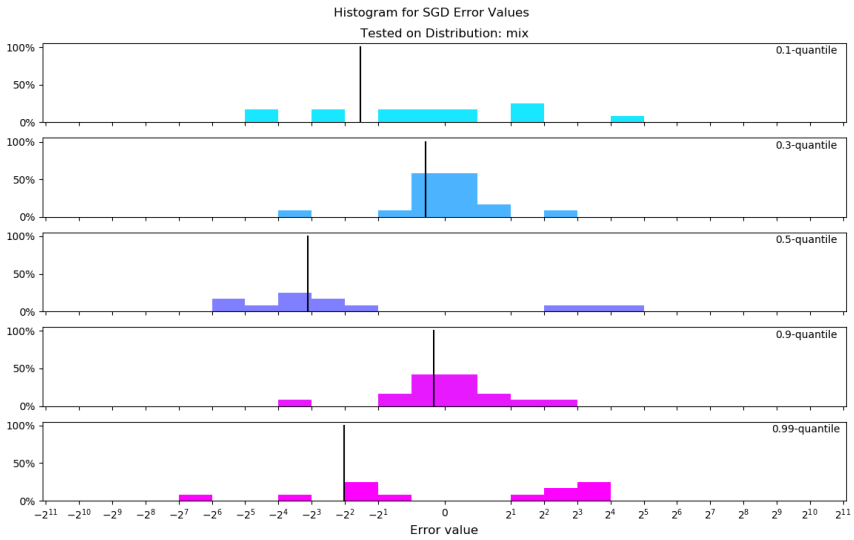

Figure 20: SGD Error from *mix* Distribution

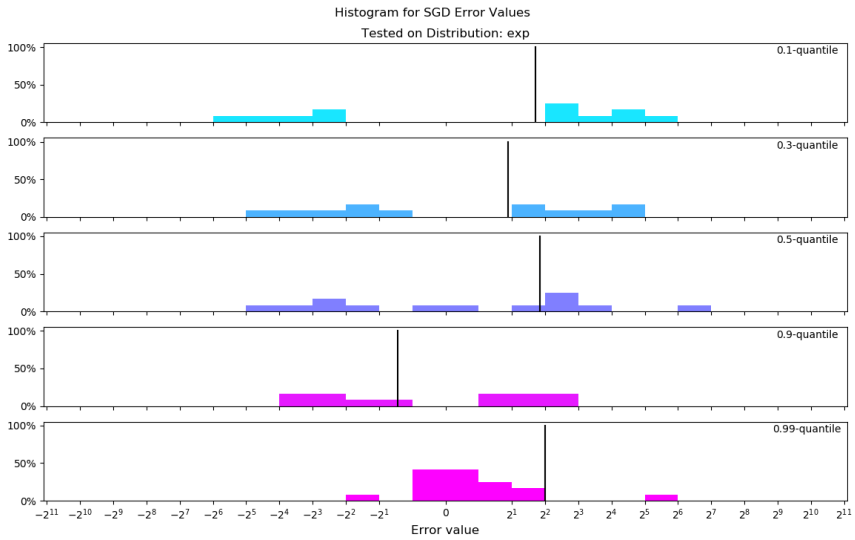

Figure 21: SGD Error from *exp* Distribution

4.3.2 *Data Size*

The experiment compares the SGD algorithm performance in 3 different data size settings: 100, 1000, 100000. In each setting, 10 data streams of the according size are generated from the *gau-1* distribution, and the SGD is run on each of the data streams. The SGD algorithm is of default setting, and the performances are evaluated on the collection of the 10 SGD implementations.

The process, result and error plots for data sizes (figure 22 to 30) lead to the following interesting observations:

1. Bigger data size leads to better convergence for all quantiles. When data size is 100, the $0.1, 0.9, 0.99$-q SGD are not converged to the true quantiles (figure 22 and 25), but when the size is 10 times bigger, the 0.1 and 0.9 quantile estimates are converged around the true values (figure 23 and 26) and the 0.99-q SGD is very close to convergence. When the data size reaches 100000, it is clear from figure 24 that all the quantiles have converged around their corresponding true values.

2. After the SGD estimates converge around the true quantiles, the fluctuation rate does not change. The "better" convergence in the last observation point refers to the stable fluctuation of quantile estimates around the true values. However, the fluctuation does not shrink after the estimate is converged. From figure 24 we can see the fluctuation of SGD quantile are not changed from the 20000th epoch to the 100000th epoch.

3. Bigger data size does not mean a better effectiveness of SGD estimation. In fact, all the error values of the size 1000 SGD quantiles are better than those of the size 100000 estimates. Specifically, the former has all its error values within range $(-2^6, 2^5)$ with mean errors in range $(-2^3, 2^3)$ (figure 29), while the latter has a much larger error distribution for every corresponding quantile estimate (figure 30). It is because the error value is calculated as a comparison with the batch quantiles. When the data size is bigger, the batch quantiles also converge to the true quantiles, while the convergence from the SGD quantiles to the batch quantiles have not changed much. Figure 26 and 27 show the batch quantiles are much denser at data size 100000, when the SGD quantiles are still distributed around them sparsely.

4.3.2.1 *Estimate Processes on Different Data Size*

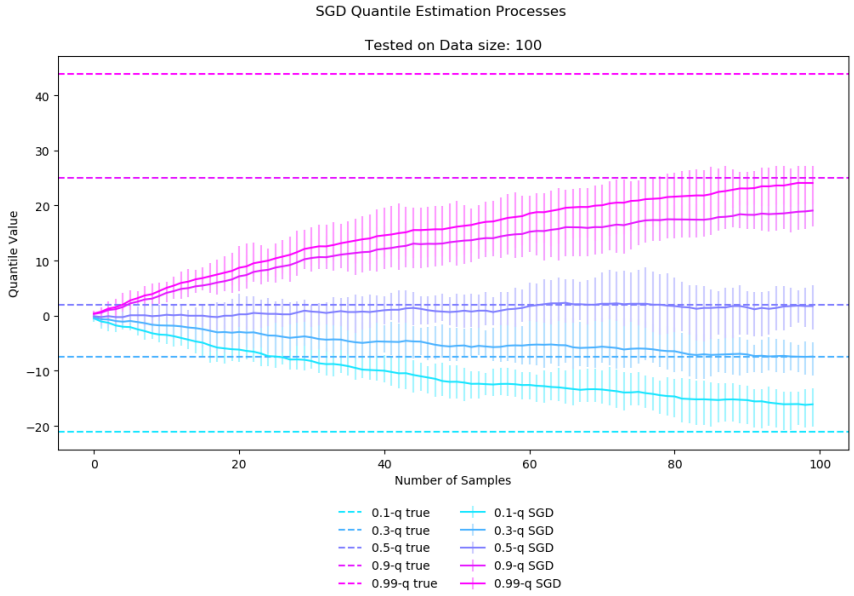

Figure 22: SGD Process from 100 Samples

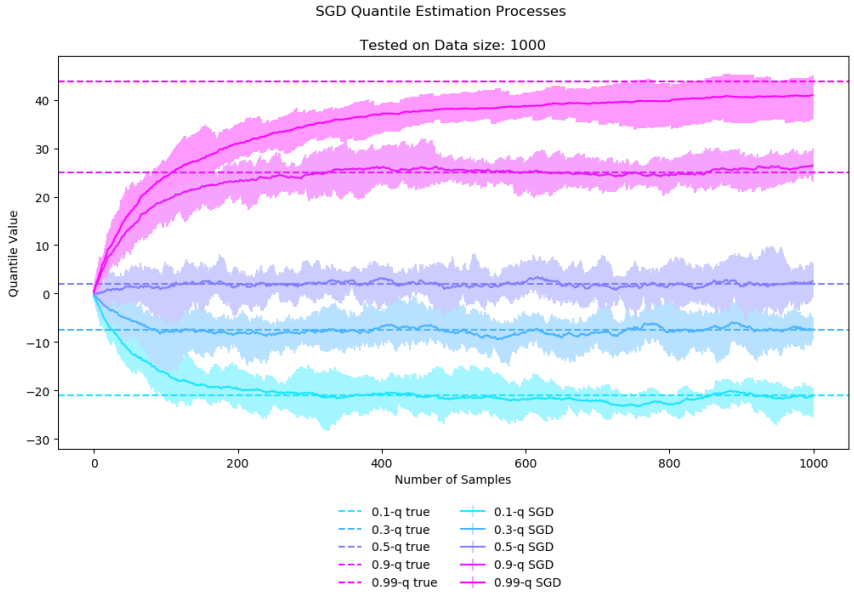

Figure 23: SGD Process from 1000 Samples

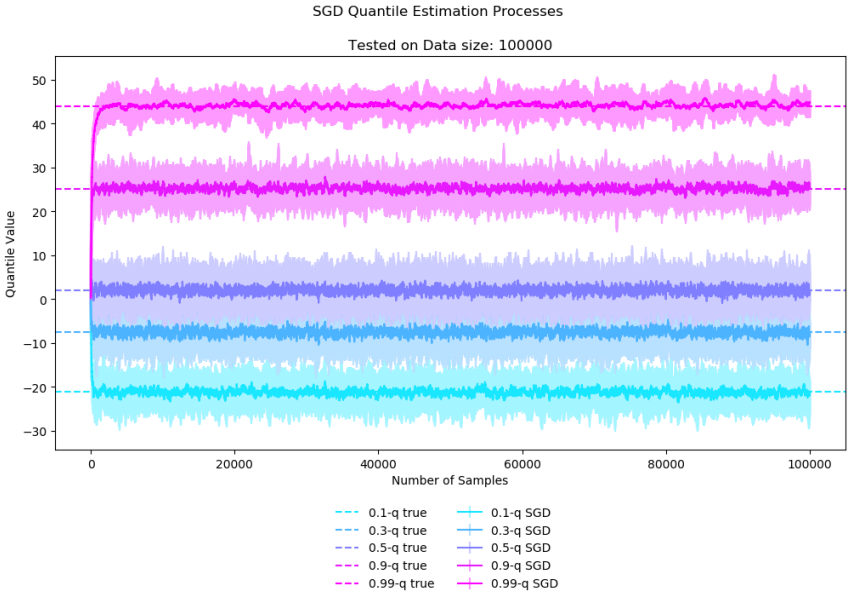

Figure 24: SGD Process from 100000 Samples

4.3.2.2 *Estimate Results on Different Data Size*

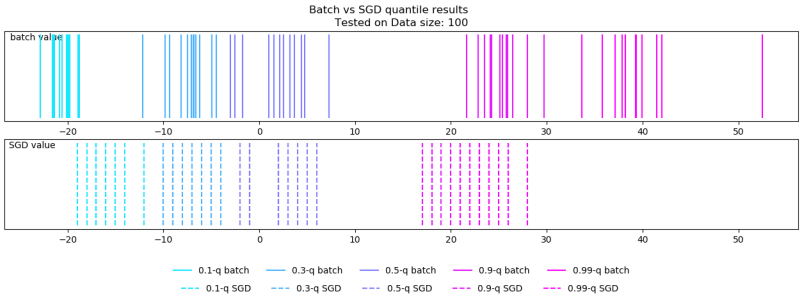

Figure 25: SGD Results from 100 Samples

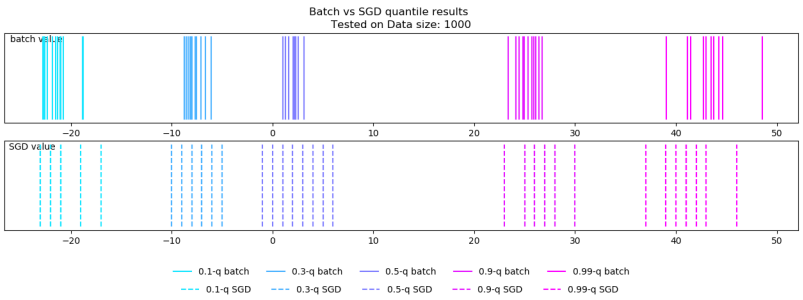

Figure 26: SGD Results from 1000 Samples

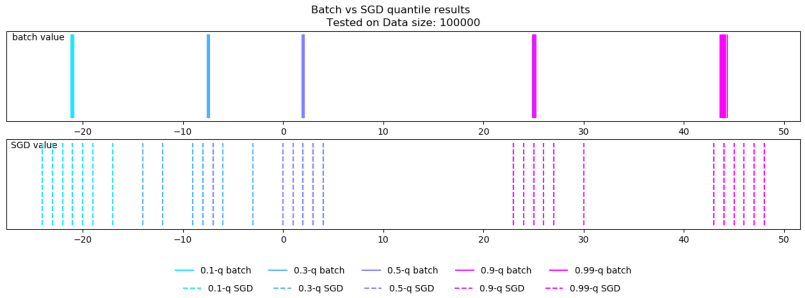

Figure 27: SGD Results from 100000 Samples

4.3.2.3 *Estimate Error Values on Different Data Size*

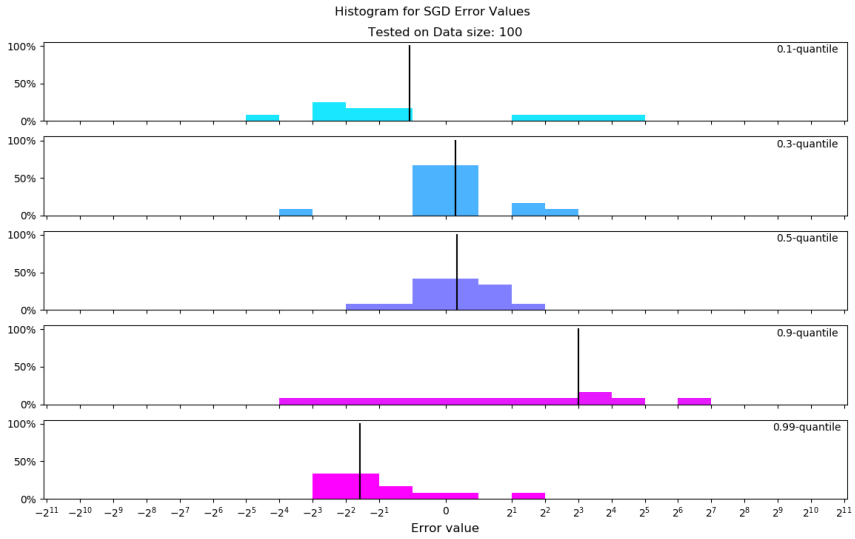

Figure 28: SGD Error from 100 Samples

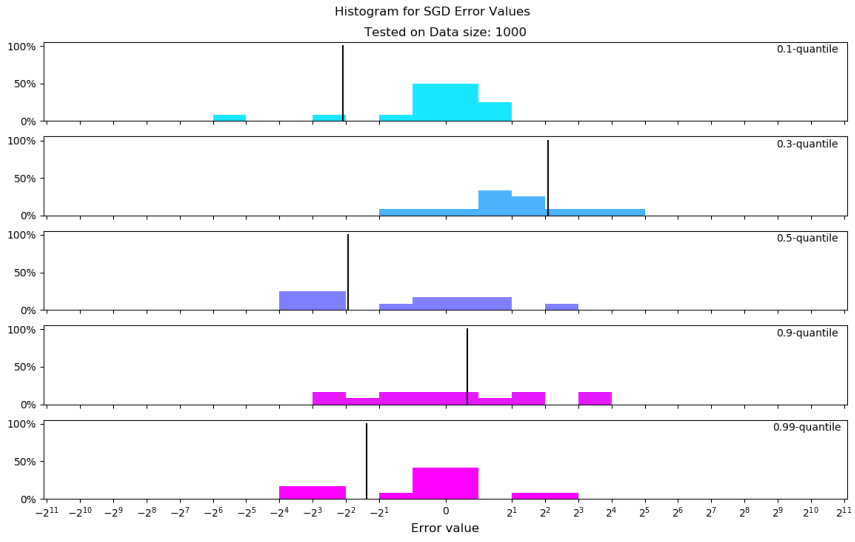

Figure 29: SGD Error from 1000 Samples

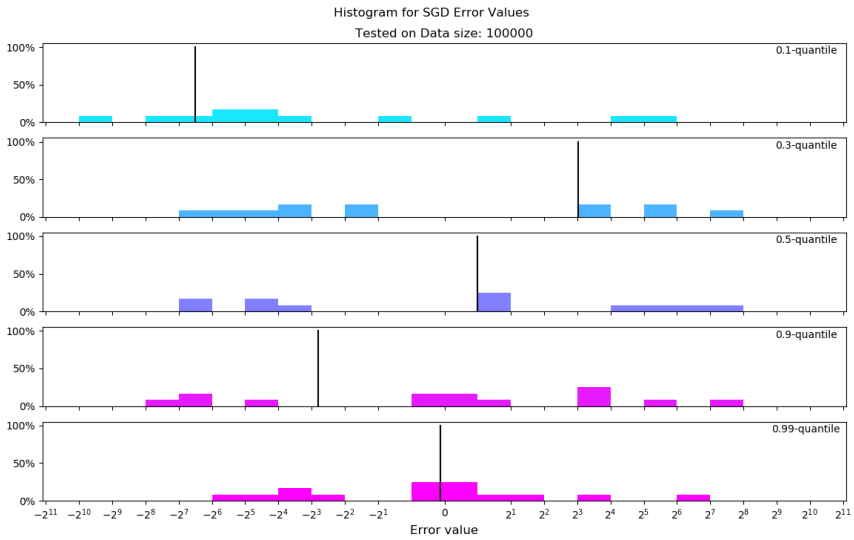

Figure 30: SGD Error from 100000 Samples

4.3.3 *Ordering of Data Stream*

The experiment compares the SGD algorithm performance when the sequence of data stream is different. Both the data streams and the SGD algorithm use the default settings. The experiment shuffles the same data stream 10 times, and illustrates the variance of SGD performances on those data streams. Note in this case, there is only one batch quantile value for each quantile.

The process, result and error plots for data sequence (figure 31 to 33) lead to the following interesting observations:

1. The impact of sequence changing is different for different quantiles. For example, the 0.99-q SGD SGD is more densely surrounding its true quantile compared with the 0.9-q SGD (see figure 32 and 33). But it is unknown if the small variance from 0.99-q SGD is temporary or lasting. For example, the variance of the 0.1-q SGD was very small during a short period at around the 900th epoch, and then it get bigger again in the end (figure 31).

2. The overall effect from changing sequence does not change much of the SGD performance. All those three plots, especially figure 33 shows that all the error values of SGD estimates are in the range $(-2^5, 2^4)$ and the mean errors are in the range $(-2^2, 2^1)$. This indicates the changes in data sequence is not likely to be a big factor of SGD performances.

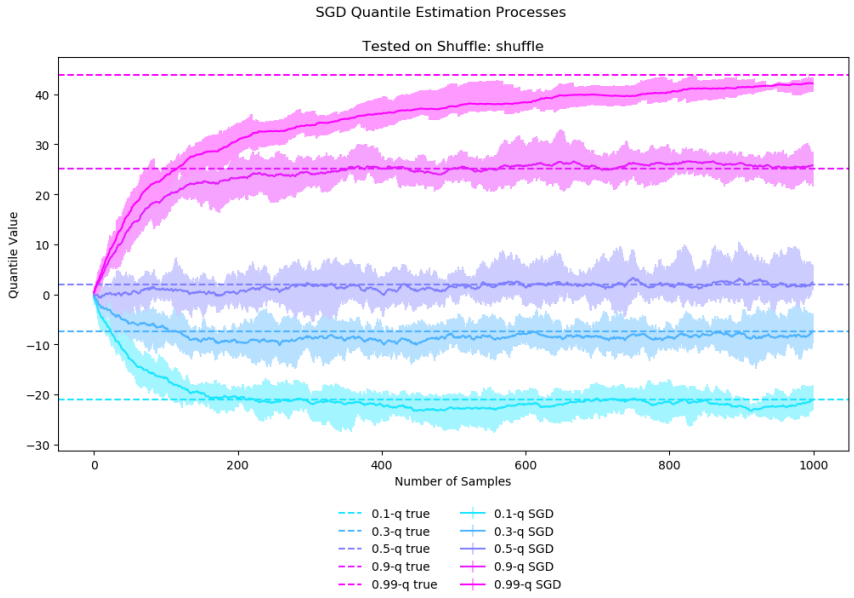

Figure 31: SGD Estimate Process from Shuffled Data Stream

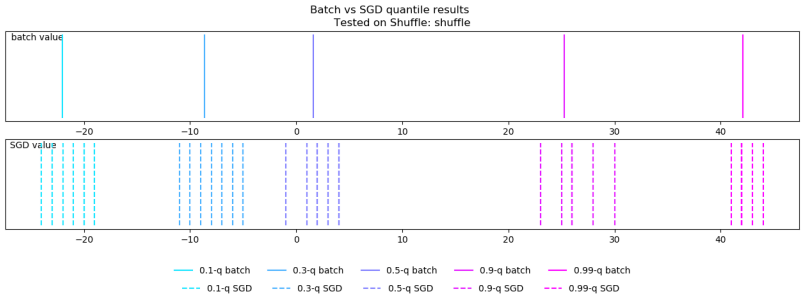

Figure 32: SGD Estimate Result from Shuffled Data Stream

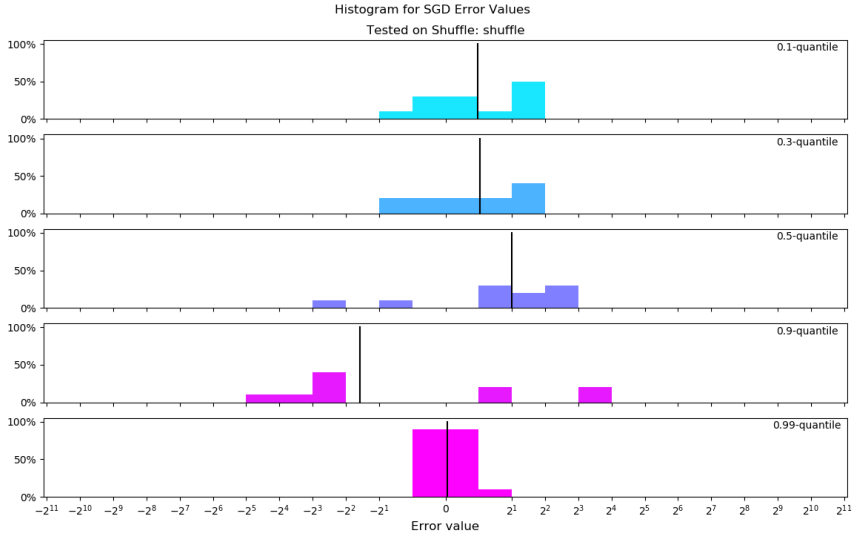

Figure 33: SGD Estimate Error from Shuffled Data Stream

4.3.4 *SGD Step Size*

The experiment compares the SGD algorithm performance on 3 different SGD step size settings: constant 1, diminishing $\alpha_i = \frac{2}{\sqrt{i}}$, or smaller diminishing $\alpha_i = \frac{0.002}{\sqrt{i}}$. For each step size setting, we generate 10 data streams with the default settings to compare the effectiveness of SGD.

The process, result and error plots for data sizes (figure 34 to 42) lead to the following interesting observations:

1. The various step size settings have vastly different behaviours on quantile estimation.

2. After convergence, the diminishing step size can reduce the fluctuating behaviour SGD exhibits. Consider the 0.5-q estimate with both a constant step size $\alpha_i = 1$ (figure 37) and a diminishing step size $\alpha_i = \frac{2}{\sqrt{i}}$ (figure 38). In both cases, the estimate converges, but with the diminishing step size it has a much smaller variance around the true quantile. This results in a smaller error value distribution around 0 (figure 41), which means a more accurate estimation.

3. The small error distribution range with mean close to 0 does not imply good convergence. When $\alpha_i = \frac{0.002}{\sqrt{i}}$, the mean error value of the 0.5-q

SGD is in $(0, 1)$, and the error discussion lies in a small range of $(-2^3, 2^5)$ (figure 42), however, it is clear in both the process and result figures (figure 36 and 39) that the SGD estimate does not move much towards the true quantile value at 2. The corresponding reflection on the error plot is that the error values are never within the range $(-2^2, 2^2)$, suggesting that the mean error is just a balance between two groups of big errors on both side.

4.3.4.1 *Estimate Processes on Different SGD Step sizes*

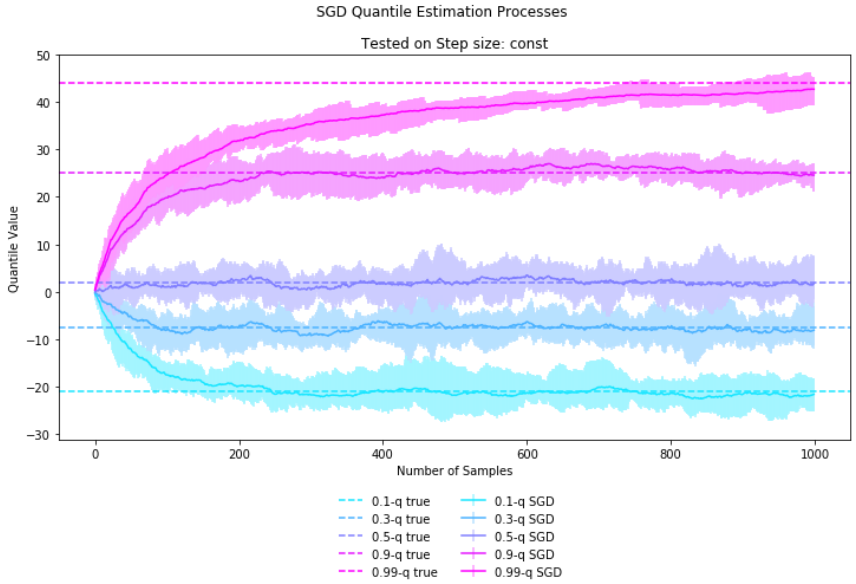

Figure 34: SGD Estimate Process from step size $\alpha_i = 1$

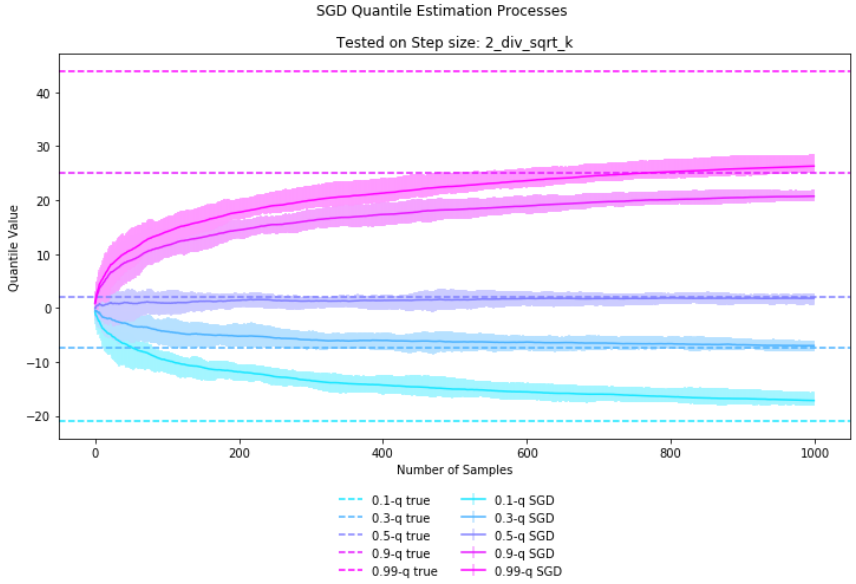

Figure 35: SGD Estimate Process from step size $\alpha_i = \frac{2}{\sqrt{i}}$

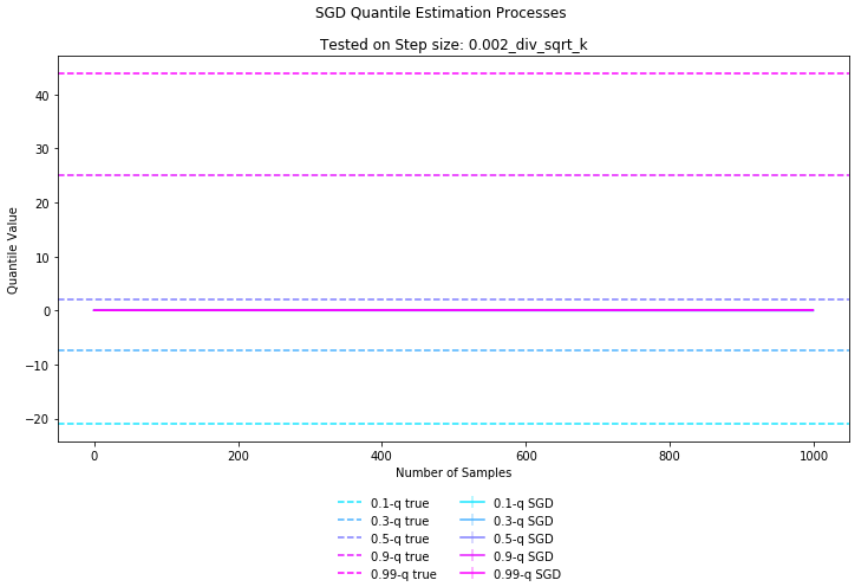

Figure 36: SGD Estimate Process from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$

4.3.4.2 *Estimate Results on Different SGD Step sizes*

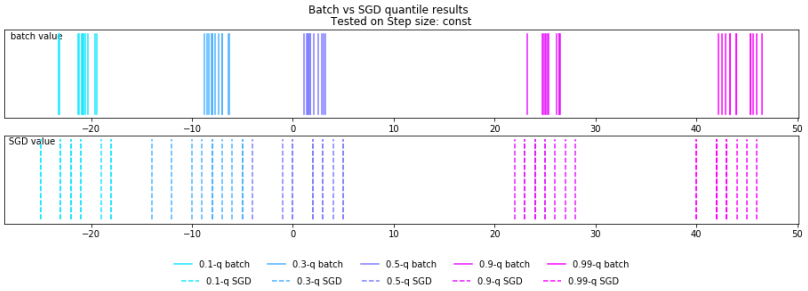

Figure 37: SGD Estimate Results from step size $\alpha_i = 1$

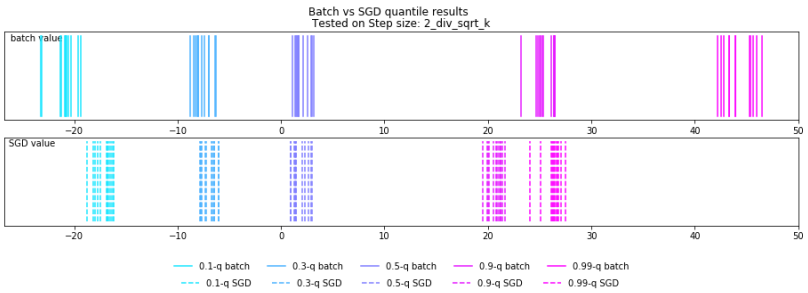

Figure 38: SGD Estimate Results from step size $\alpha_i = \frac{2}{\sqrt{i}}$

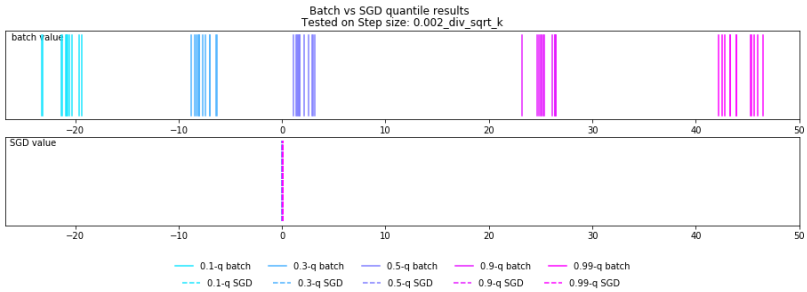

Figure 39: SGD Estimate Results from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$

4.3.4.3 *Estimate Error Values on Different SGD Step sizes*

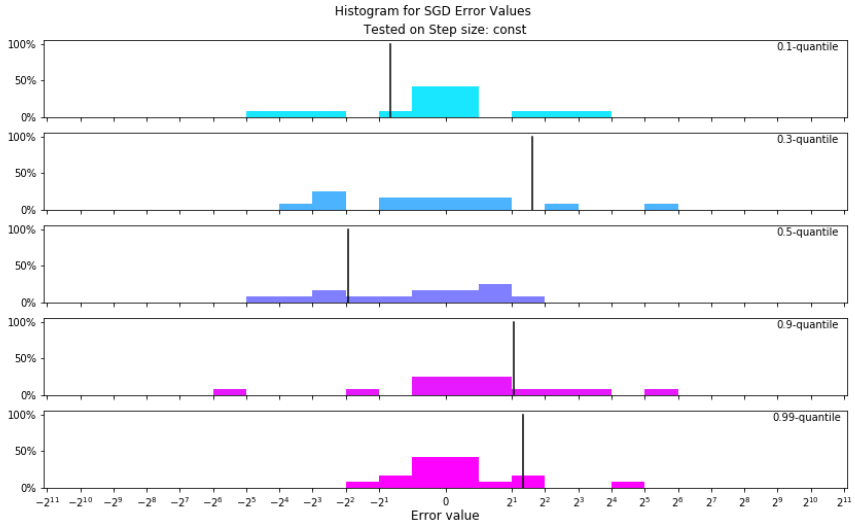

Figure 40: SGD Estimate Error from step size $\alpha_i = 1$

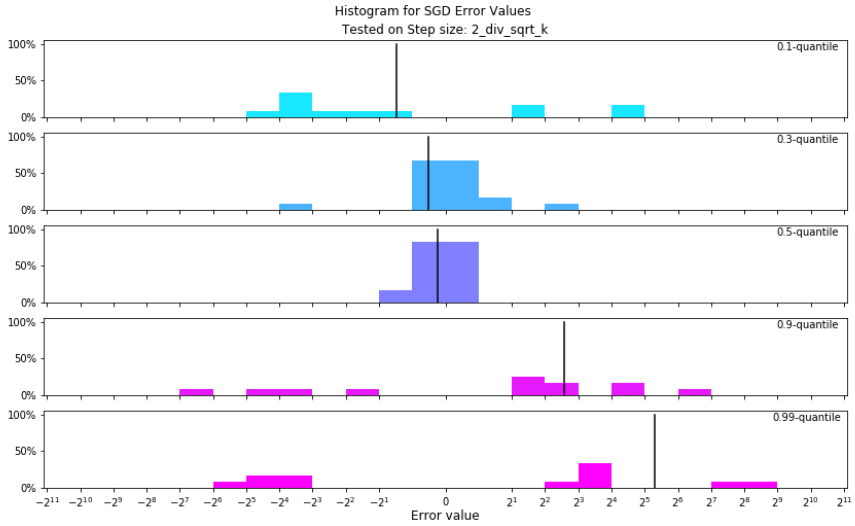

Figure 41: SGD Estimate Error from step size $\alpha_i = \frac{2}{\sqrt{i}}$

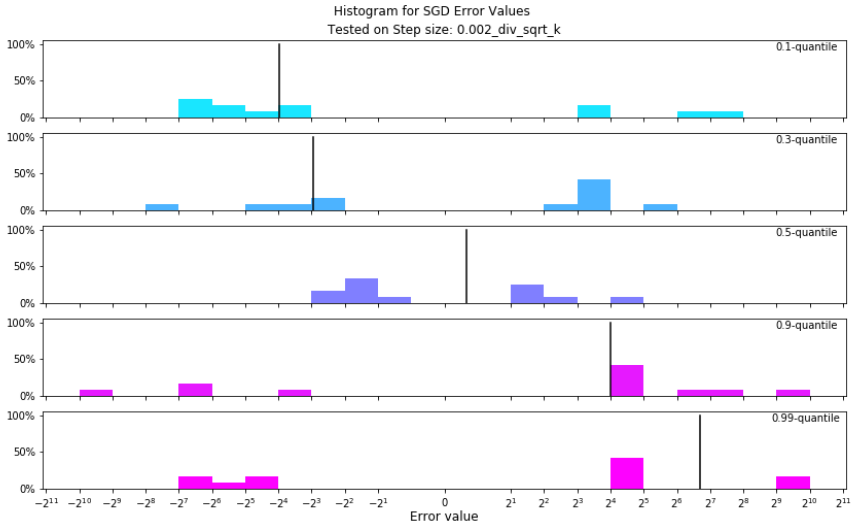

Figure 42: SGD Estimate Error from step size $\alpha_i = \frac{0.002}{\sqrt{i}}$

4.4 DISCUSSION

4.4.1 *SGD step sizes and distributions*

The selection of appropriate step sizes for different distributions is of great importance. The impact of excessively small step sizes on a distribution is shown in section 4.3.4. The extreme case with overly big step size, on the other hand, is illustrated from the weird performance of SGD in the *gau-2* distributions (see subsection 4.3.1).

The uncommon performance of it is reflected in all the three aspects of process, results and error values. For example, we notice the regular sawtooth update line of the 0.99-q SGD in figure 11. The mechanism of 0.99-q SGD is that it increase by 0.99 when observe a data greater than the current estimate, and decrease by 0.01 otherwise. Note in *gau-2*, the probability of sampling a data outside $(-0.01, 0.01)$ is close to 0. This means, the initialization estimate at 0 is highly likely to lead to an increase of 0.99 in the first or the second epoch. Once the quantile estimate is beyond the range $(-0.01, 0.01)$, the next data is nearly impossible to be a bigger one, so the estimate decreases with possibility close to 1. Which is why the 0.99-q SGD decrease by 0.01 until it reaches 0 again. The round of "start from 0, (maybe decrease a bit,) increase for one iteration, then decrease back to 0" takes exactly 100 epochs. So the results of the 0.99-q SGD nearly always ends up at 0 after 1000 epochs. The other quantiles face with the similar problem, so they are also very likely to reach exactly 0 after 1000 epochs. This explains why in figure 15 all SGD estimates are in the same value 0.

The choice of step size is not only important for SGD on quantile estimation, but is also a popular topic for general SGD methods in machine learning. In chapter 5, we provide some further discussions on the adaptation of step size for different data streams inspired by the current machine learning methods.

4.4.2 *SGD step size and data size*

This paper does not include any experiment investigating the relationship between SGD step size and dataset size, but the process figure of step size $\alpha_i = \frac{2}{\sqrt{i}}$ (figure 35) shows the trend of 0.9 and 0.1-q SGD are approaching their true quantile, and it is likely that the convergence can be reached in another 1000 epochs. For the 0.99-q SGD however, will likely take much more than another 1000 epochs to converge, as it is further away from the true quantile and the slope of the trend is low.

The convergence issue of step size $\alpha_i = \frac{0.002}{\sqrt{i}}$ can hardly be solved by the expansion of data stream size. From figure 36 we can see the slow SGD update trend for all quantiles are almost flat. Under the condition that the step sizes are diminishing, it highly possible that the convergence would not happen for any quantiles even if the data is 100 times bigger.

4.4.3 *Starting point and the convergence*

The starting point of SGD is another interesting variable of the SGD algorithm, but it is not investigated in this chapter. For example, in the experiment that the SGD step size is $\alpha_i = \frac{0.002}{\sqrt{i}}$ for the $gau-1$ distribution, if the initialization of

the 0.5-q SGD starts at 1.9 instead of 0 (the true 0.5-q = 2), then the SGD will have a much more accurate estimation.

An optimal choice of initialization is obviously the true quantile, however this is obviously infeasible. A feasible option is to have the initial quantile estimate at the position close to the true quantile. One possible solution would be to sort the first few data points and use the batch quantile of this group of small samples as initial estimates. This is a potential further improvement direction for our SGD quantile estimation research.

4.5 CONCLUSION

In this chapter, we conducted experiments on SGD for quantile estimation. We find that the performance of the algorithm is affected by the input data stream as well as the variable settings in SGD. For data streams, the changes in data distribution, data size and data sequence all affect the performances of SGD. Specifically, the SGD works well when

- The true quantiles in the data distribution are well-distributed. If they are too far from the initial quantile estimate, the quantile estimates will take too many epochs to converge (e.g., the 0.99-q for *gau-1*). At the same time, the true quantiles should not be too close to each other. Otherwise the quantile estimates are likely to violate the monotone property, and additionally the low precision can be pointless for the estimation result (e.g., all quantiles in *gau-2*).
- The data size is not too small for the SGD quantiles to converge (e.g., the 0.99-q for data size = 100), and at the same time not so big that the precision of SGD estimation is close to the batch quantiles (e.g., all quantiles for data size = 100000)
- The step size of SGD is big enough to converge, and once it has converged the step size continues to decrease, reducing fluctuation and variance (e.g., the 0.3-q for diminishing step size $\alpha_i = \frac{2}{\sqrt{i}}$).

The experiments found that the ordering of the data sequence does not have a big impact on the performance, but this conclusion needs to be checked with more empirical experiments or mathematical explanations. Besides these observations, we believe the combination of various settings can greatly improve performance. For example, the the slow convergence caused by a small step size is not an issue with a much larger data stream. It is also mentioned that the initialization of SGD quantiles can be another contributing variable for the estimation performance.

Since the properties of the input data stream are unknown to the algorithm, the only improvement direction for SGD is to change its settings so it works better in different conditions. In the following chapters, we explore the improvement of SGD on two problems: the crossing quantile estimates and the adaptation of step size. In chapter 6 we look into potential solutions to keep the monotone properties, and in chapter 5 we introduce our new method for step size adaptation.

5 STEP SIZE ADAPTATION

Step size is a key factor in convergence performance. The importance of step size selection is not only shown in our practical experiment results (chapter 4), but has also been proved in theories of the machine learning ares. Generally in SGD, a large step size is likely to converge quickly and then have a high fluctuation rate. In contrast, a small step size can reduce fluctuation after it eventually converges, but the data stream might stop before convergence is reached. Due to the fact that a small step size for one data stream might be a big one for another, we want to choose step sizes based on different measurements and heuristics. That is, we want to explore the topic of step size adaptation.

In this chapter, we conduct an investigation of step size adaptation for SGD quantile estimation. Specifically, we look into two potential machine learning methods on step size adaptation, make analysis and implement one of them on SGD. In addition, we propose another step size adaptation algorithm, and compare the difference between their empirical performances. The contents of this chapther is listed as follows:

Section 5.1 briefly introduces the potential step size adaptation method in machine learning, and explains why it cannot be applied for the SGD algorithm.

Section 5.2 analyses another machine learning method *SAG*, implements it to SGD and conducts experiments to check the convergence performance

Section 5.3 shows a proposal of another step size adaptation algorithm *DB-SGD*, which is also provided with explanations and empirical experiments.

Section 5.4 compares the two proposed algorithms and discusses about their improvements on convergence.

5.1 FAILURE ON NEWTON'S METHOD

Newton's method is an iterative method to find the stationary points (where the function's derivative is zero) of a twice-differentiable function f. The application of Newton's method failed for our SGD quantile methods, since the loss function of the quantile estimation function is not twice-differentiable. For a specific τ, let $t := x - q$ be the difference between the input data value x and the estimate of quantile q, the loss function

$$\ell_\tau(t) = \begin{cases} \tau t & t > 0 \\ -(1-\tau)t & otherwise \end{cases} \tag{21}$$

is a linear function of t, which doesn't have any second derivative.

Though the method cannot be applied, it is easy to reach the goal of Newton's method: to find the critical points of a function. Instead of stationary point, the loss function has a critical point where it is not differentiable and the derivative changes sign. For any $\tau \in (0,1)$, when $t = 0$, the loss function reaches it's critical points at $\ell_\tau(0) = 0$. Taking the critical point for each observation, however,

does not contribute to any improvement in quantile estimation. To be at a critical point, the quantile estimate is set to have the equal value of input data x, and only in this way we could have $t = x - q = x - x = 0$. Regardless of τ, the quantile estimate is always equal to the value of the latest data point. Hence, this method completely fails its goal to estimate a quantile value based on τ and the entire data stream.

From another perspective, the failure of Newton's method is the result of applying large step size for the last input data for a SGD method. In this way, the minimal of current loss function $\ell_\tau(t)$ is reached, while the total loss function for the input data stream X

$$L_\tau(t) = \sum_{x \in X} \ell_\tau(t) \tag{22}$$

is entirely ignored.

5.2 STOCHASTIC AVERAGE GRADIENT (SAG)

One approach on step size adaptation is to use the stochastic average gradient (SAG)[29] algorithm. It is an convex optimization method that has a significant improvement of convergence rate than stochastic gradient (SG) methods. In general, the convergence rate is improved from $O(1/\sqrt{k})$ to $O(1/k)$ for a total of k epochs, reaching the same level as the gradient descent method. Along with the improved convergence rate, the computation time for each iteration is independent of the size of function sum.

5.2.1 *Mechanism of SAG*

Recall the update function of SGD (equation 7), the stochastic average gradient (SAG) remembers the last update of a gradient value for each index i, which enables the improvement of convergence rate from SG methods. Its iteration takes the form

$$x_{i+1} = x_i - \frac{\alpha_i}{N} \sum_{n=1}^{N} y_m^i \tag{23}$$

where y_m^i is used to keep the memory of recently updated gradient value of function ℓ_m

$$y_m^i := \begin{cases} \ell'_m(x_i) & \text{if } m = m_i \\ y_m^{i-1} & \text{otherwise} \end{cases} \tag{24}$$

In reference to the SAG algorithm, Schmidt, Roux, and Bach state that "like the FG method, the step incorporates a gradient with respect to each function". Meanwhile only one gradient computation is involved in the combination of gradients, meaning that the iteration cost is independent of N.

5.2.1.1 *Convergence guarantee*

The SAG algorithm with constant step size $\alpha_i = \frac{1}{16L}$ reaches the convergence rate of $O(1/i)$ under the certain assumptions. Each ℓ_m must be convex and the gradient ℓ'_m must be *Lipschitz continuous* with constant L, that is

$$|\ell'_m(a) - \ell'_m(b)| \leqslant L|a - b| \tag{25}$$

for all $a, b \in \mathbb{R}^p$. The convergence function differs when y_m^0 is initialized differently, or when ℓ_m is strongly convex.

5.2.2 *Basic SAG algorithm*

Algorithm 4 Basic SAG method for minimizing $\frac{1}{N}\sum_{n=1}^{N}\ell_n(x)$ with step size α

Input: Dataset X, Dataset Size N, Step size α
Output: x

```
d = 0, y_m = 0 for m = 1, 2, ..., N        ▷ Default initialization
for i = 0, 1, ... do
    Sample m from {1, 2, ..., N}
    d = d − y_m + ℓ'_m(x)
    y_m = ℓ'_m(x)                          ▷ Save the y_m in the table for re-visit
    x = x − (α/N) d
end for
```

The basic SAG algorithm requires memory storage of a table of $y_m (m = 1, 2, ..., N)$, to keep the track of each y_m in case they are re-visited after their first update.

5.2.3 *SAG Implementation for quantile estimation*

The quantile estimation loss function is a convex function, and we can use a smooth function for replacemer
The step size $\alpha = \frac{1}{16L}$ guarantees the convergence of SAG, where L is the Lips-

Algorithm 5 Basic SAG method for streaming data S for quantile estimation

Input: Data Stream X, Data Stream Size N, τ, τ-quantile estimate q, Step size α
Output: q

```
Initialization d = 0, q = 0        ▷ Default initialization q_0 = 0, d_0 = 0
for each x_i in X do
    d = d − 0 + ℓ'_τ(x_i, q)       ▷ 0 stands for y_m^{i−1}
    q = q − (α/N) d
end for
```

chitz constant for the pinball loss function of τ. Here we have $L = \min(|\tau|, |1 - \tau|)$. The computation of L is discussed in subsection 5.2.5. For streaming data, it's worthwhile noticing that each input data point has exactly one pass. This means for the SAG implementation, the storage of updated y_m^i is useless since it would not be revisited. Besides, if all y_m^0 are initialized as 0, the storage of y^0 initialization needs only one unit of memory instead of N units of memory. in this way, we can keep the memory complexity of SAG quantile estimation to $O(1)$.

5.2.4 *Experiments on SAG*

In this part, we explore how SAG improves the performance of SGD convergence. The results shown in plots of SAG (figure 43, 44 and 45) has led to the following observations:

1. The convergence of SAG is faster than the SGD algorithm for some quantiles. From figure 43 we can see the number of epochs to reach the convergence for 0.99-q is improved from more than 1000 to less than 200, and the number of epochs for 0.1-q SAG and 0.9-q SAG are around the same as SGD, while SAG convergence for 0.3-q SAG is significantly slower.

2. The fluctuation around different quantile are different. It is shown in figure 43 that the 0.99-q SAG has a much larger range of fluctuation around the true quantile than that of 0.5-q SAG.

3. The overall estimation accuracy of SAG is relatively acceptable. Among the trails of 50 estimations, 49 of them are in the error value range $(-2^6, 2^4)$, and the only one outlier is at 2^7. The mean of the error values are also within $(-2^2, 2^3)$, meaning the expectation of all estimates is close to the batch values.

4. It is shown in figure 44 that the 0.99-q ends up with only two possible result values. The reason of this is unknown, but one possible cause is that $\alpha_{0.99}$ is too big for this quantile so it has not much choice of changing values (similar to SGD on *gau-2* results). Also given there were only 10 experiments, it also might be a coincidence. More experiment data is required for this observation.

It is worthwhile to notice that the difference in convergence rate and fluctuation rate for different quantiles are caused by their different step size scaler α_τ. As $\alpha_\tau = \frac{1}{16L}$ and $L = \min(|\tau|, |1-\tau|)$, it means $\alpha_{0.99} = 25/4$ while $\alpha_{0.9} = 5/8$. Now that $\alpha_{0.99}$ is 10 times the size of $\alpha_{0.5}$, it is reasonable for 0.99-q SAG to converge faster than 0.9-q SAG. It also explains why the fluctuation for 0.99-q is much larger than the other quantiles.

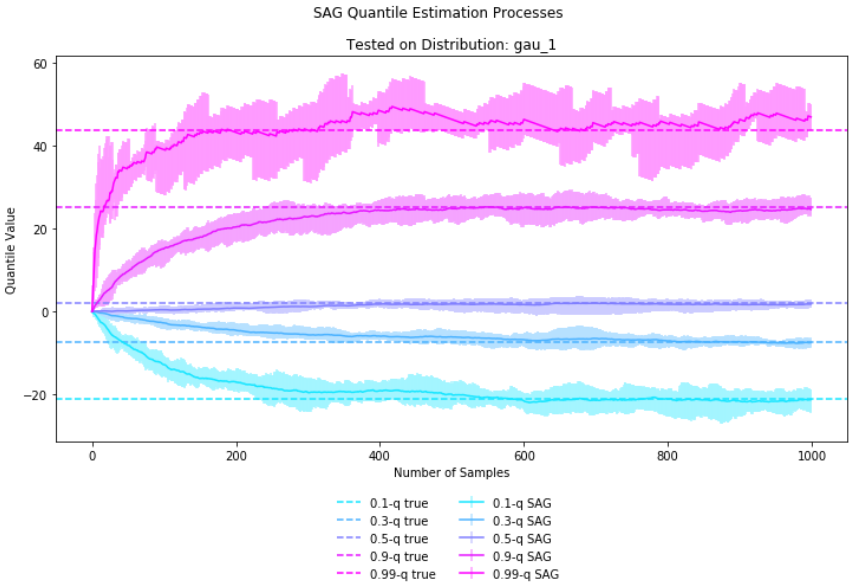

Figure 43: SAG Process from *gau-1* Distribution

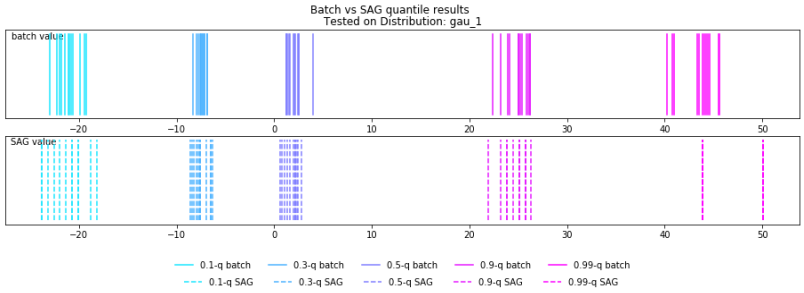

Figure 44: SAG Result from *gau-1* Distribution

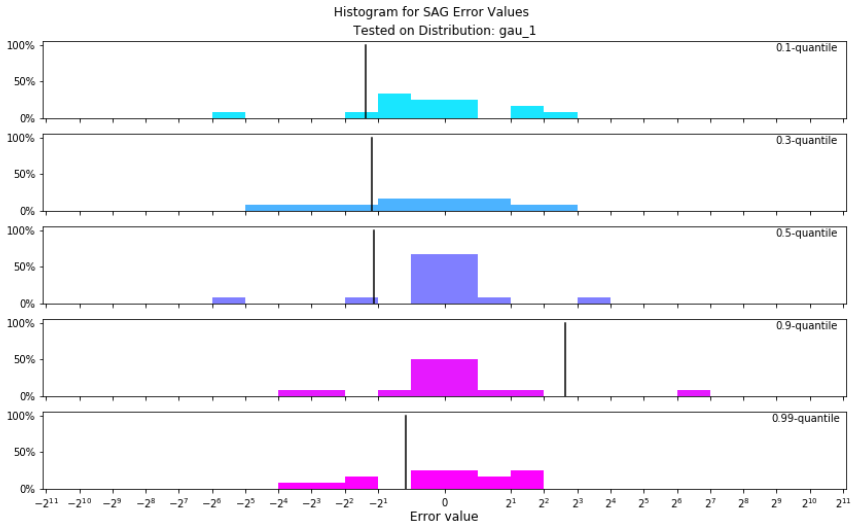

Figure 45: SAG Error from *gau-1* Distribution

5.2.5 *Smooth Functions*

The pinball loss function we use for the SGD quantile estimation is convex but not smooth. Theoretically, however, both SGD and SAG methods need smoothness for the guarantee of convergence. Although the practical experiments have shown that there is evidence of convergence in final performance, it remains a serious problem for our SGD quantile estimation methods. In this section, we present the analysis on the non-smoothness problem, followed by the potential solutions and further discussions on them.

5.2.5.1 *The Pinball Loss Functions*

The pinball loss function evaluates the loss from a data point x for estimation of the τ-quantile.

The fact that the pinball loss function is not differentiable at $x = 0$ is a problem for the stochastic gradient descent which uses the derivative of the objective function.

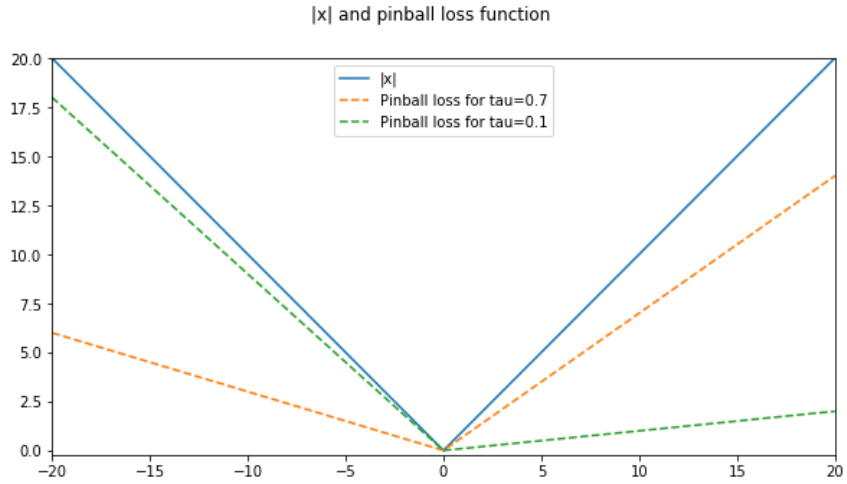

Figure 46: Comparison between $|x|$ and the pinball loss function with different τ values

The absolute value function and the pinball loss function are obviously very similar. In fact, with simple manipulation, the pinball loss function can be written in the form of absolute function as

$$l_\tau(x) = \begin{cases} \tau \cdot |x| & x \geqslant 0 \\ (1-\tau) \cdot |x| & \text{otherwise} \end{cases} \tag{26}$$

The pinball loss function is the combination of $|x|$ on the two parts $x < 0$ and $x > 0$ with different constant scale. If the derivative of the smooth function for $|x|$ at $x = 0$ is 0, we can also use the same approximation function respectively for the two parts of the pinball loss.

In section 5.2.5.2 and 5.2.5.3, we show two approximation functions with regards to the absolute function $f = |x|$, along with the application of the approximations on the pinball loss function.

5.2.5.2 *Smooth function 1: a simple approxitmation*

For $f = |x|$, a common and simple smooth approximation is $f = \sqrt{x^2 + \mu^2}$. The convergence towards the absolute function is then proved by Voronin, Ozkaya, and Yoshida[32] that

$$||x| - \sqrt{x^2 + \mu^2}| \leqslant \mu \text{ where } \mu > 0 \in \mathbb{R} \tag{27}$$

The smooth function application on the pinball loss function is:

$$l_\tau(x) \approx \begin{cases} \tau \cdot \sqrt{x^2 + \mu^2} & x \geqslant 0 \\ (1-\tau) \cdot \sqrt{x^2 + \mu^2} & x < 0 \end{cases} \tag{28}$$

and the derivative is

$$l_\tau\prime(x) \approx \begin{cases} \tau \cdot \frac{x}{\sqrt{x^2+\mu^2}} & x > 0 \\ 0 & x = 0 \\ (1-\tau) \cdot \frac{x}{\sqrt{x^2+\mu^2}} & x < 0 \end{cases} \tag{29}$$

5.2.5.3 *Smooth function 2: a transcendental approximation*

For a better approximation accuracy, Bagul[3] proposes a new transcendental approximation function

$$g(x) = x \cdot \tanh(x/\mu) \text{ where } \mu > 0 \in \mathbb{R} \tag{30}$$

which also satisfies

$$||x| - x \cdot \tanh(\frac{x}{\mu})| < \mu \tag{31}$$

The derivative of the approximation is

$$\frac{dg(x)}{dx} = \frac{x}{\mu} \cdot \operatorname{sech}^2(\frac{x}{\mu}) + \tanh(\frac{x}{\mu}) \tag{32}$$

For the pinball loss function, we now have a smooth approximation of the form:

$$l_\tau(x) \approx \begin{cases} \tau \cdot x \cdot \tanh(x/\mu) & x \geqslant 0 \\ (1-\tau) \cdot x \cdot \tanh(x/\mu) & x < 0 \end{cases} \tag{33}$$

and the derivative is

$$l_\tau\prime(x) \approx \begin{cases} \tau \cdot \frac{x}{\mu} \cdot \operatorname{sech}^2(\frac{x}{\mu}) + \tanh(\frac{x}{\mu}) & x \geqslant 0 \\ 0 & x = 0 \\ (1-\tau) \cdot \frac{x}{\mu} \cdot \operatorname{sech}^2(\frac{x}{\mu}) + \tanh(\frac{x}{\mu}) & x < 0 \end{cases} \tag{34}$$

5.2.5.4 *Discussion and Conclusion*

In this part, we will compare the two alternative smooth functions in the following aspects: computer efficiency, approximation accuracy and convexity.

- Computation efficiency:
  It is proved by Ramirez et al.[28] that $f = \sqrt{x^2 + \mu^2}$ is the most computationally efficient smooth approximation to $|x|$.

  On the other hand, we can see from equation 34, the derivative for $g(x) = x \cdot \tanh(x/\mu)$ is more difficult to compute.

- Approximation accuracy:
  The following plots show intuitively how fast the two smooth functions approach to $|x|$ for $\mu = 0.01$ and $\mu = 0.001$.

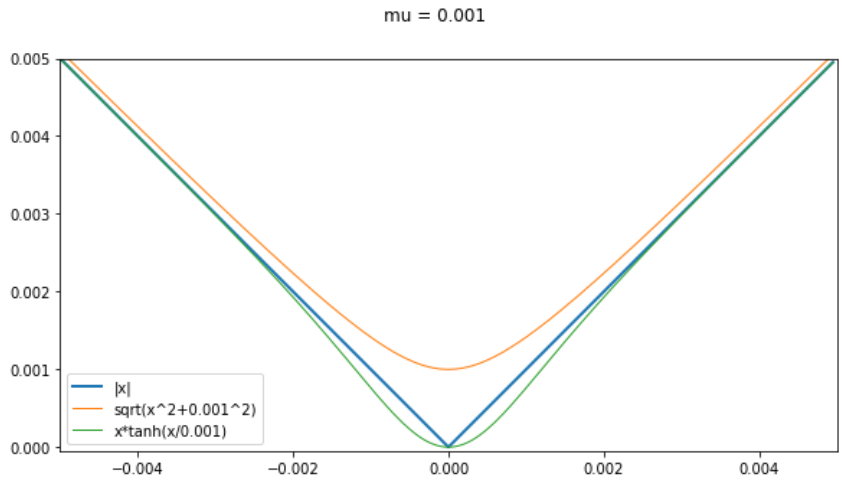

Figure 47: Comparison between the two smooth functions when $\mu = 0.001$

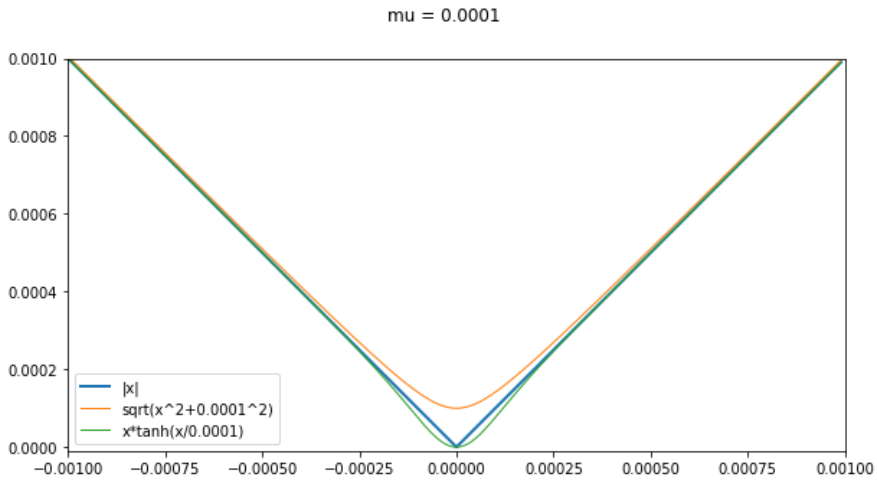

Figure 48: Comparison between the two smooth functions when $\mu = 0.0001$

It is obvious by inspection that $x \cdot \tanh(x/\mu)$ has a better accuracy.

- Convexity
  $f = \sqrt{x^2 + \mu^2}$ is convex and $g(x) = x \cdot \tanh(x/\mu)$ is not.

The form of smooth function does not really matter for SAG. It is important to know the existence of Lipschitz continuous smooth function, and then the L of such function must satisfy

$$L \leqslant \min(|\tau|, |1-\tau|) \tag{35}$$

So that no matter which smooth function is used, we can simply take $\min(|\tau|, |1-\tau|)$ to compute the step size α for SAG. However, we can still implement those smooth functions for SGD. Further research can be done on different smooth function implementations for SGD.

5.3 DOUBLING AND HALVING SGD (DH-SGD)

Although SAG has a faster convergence rate than SGD, it is clear that the fluctuation problem after convergence still exists. Is it possible that a step size adaptation of SGD can reduce the fluctuation after convergence, and at the same time have an improved convergence rate? In this section, we propose a simple algorithm called Doubling and Halving Stochastic Gradient Descent (DH-SGD), which empirically mitigates those two problems.

5.3.1 *Method Description*

The idea of the DH-SGD method is trivial - increase the step size if it is too big, and decrease the step size if it is too small. We choose the change of step sizes to be exponential, specifically, increasing it to double size or decrease it to half size. However, the standard on the scale of step size can be vague.

Here we use an intuitive idea which tracks the proportion of increase and decrease updates among an interval of updates. For example, for every 200 epochs, the proportion of increase updates is P^+, and that of decrease update is P^-, we compute $P = P^+ - P^-$ as the proportional difference between the two update directions. If P is too far from the ideal value P^* ($\mathbb{E}(P)$ for converged quantile estimate), then it means the updates are mostly in one direction, which is likely caused when the quantile hasn't yet converged, meaning that the step size is too small. Otherwise if P is close to P^*, it is a sign that convergence has already been reached, and reducing the step size will reduce the fluctuation.

According to the SGD update, the current quantile estimate decrease when it sees an data point smaller than it, and increases on seeing a larger one. This means when the quantile estimate for τ-quantile has converged, the proportion of increase update P_τ^+ should be the proportion of data points bigger than it, which is $1-\tau$. Similarly, we have the corresponding expectation that $P_\tau^- = \tau$. Thus we have the ideal value for quantile probability τ defined as

$$P_\tau^* = \mathbb{E}(P_\tau) = \mathbb{E}(P_\tau^+) - \mathbb{E}(P_\tau^-) = (1-\tau) - \tau = 1 - 2\tau \in (-1, 1) \tag{36}$$

We use Euclidean distance to measure how close P_τ is to P_τ^*. If $P_\tau \in (P_\tau^* - 0.01, P_\tau^* + 0.01)$, it means they are close and the step size is too big. And if $P_\tau \notin (P_\tau^* - 0.1, P_\tau^* + 0.1)$, it means they are too far away and the step size is too small. The psedocode of DH-SGD is shown as algorithm 6 :

Algorithm 6 DH-SGD algorithm

Input: Data Stream X, τ, 1 unit of memory q, epoch check size c
Output: q

idealProportion $P^* = 1 - 2\tau$
epochCount $ec = 0$ ▷ The counter for epochs
directionCount $dc = 0$ ▷ The accumulation of directions
scaler $s = 1$ ▷ The scaler for step size
Initialize q ▷ Default initialization $q_0 = 0$

for x_i in X **do** ▷ Parameter update for each input data point

    $ec = ec + 1$
    if ec *mod* $c = 0$ **then** ▷ Change scaler for every c epochs
        if $\frac{dc}{c} \in (P^* - 0.01, P^* + 0.01)$ **then**
            $s = 2s$ ▷ Double s if the update is mostly in one direction
        elseif $\frac{dc}{c} \notin (P^* - 0.1, P^* + 0.1)$ **then**
            $s = 0.5s$ ▷ Halve s if the update direction barely changes

    set $\alpha_i = 1$ ▷ α_i can use other settings
    $\alpha_i = \alpha_i \cdot s$ ▷ Scale original step size with s
    if $x_i > q$ **then**
        $q = q + \alpha_i \tau$
        $dc = dc + 1$ ▷ Direction count $+1$ if updated upwards
    elseif $x_i < q$ **then**
        $q = q - \alpha_i (1 - \tau)$
        $dc = dc - 1$ ▷ Direction count -1 if updated downwards

return q

5.3.2 *Experiments*

Again we test the DH-SGD on the data stream of size 1000 of *gau-1* distribution. The experiment results are shown in figure 49, 50 and 51. Interesting Observations:

1. The change of step size has a very obvious improvement on convergence rate. From figure 49 we can see all the quantiles has their convergence rate at the same or faster rate compared with SGD. For example, both 0.99 and 0.9-q DH-SGD have a doubled step size at epoch 200, and the former reaches its convergence around epoch 900 (compared with more than 1000 in SGD) and the latter reaches convergence within 300 epochs (compared with around 600 in SGD). The other three quantiles have basically the same convergence rate.

2. The fluctuation after convergence are also improved overall. All the quantiles except for the 0.1-q DH-SGD has a denser results distribution around the batch results (figure 50). However, there are also results that stand out from nearly every estimation distribution, which means the reduction of fluctuation is not consistent.

3. The effect on error value is a combined result from denser distribution of accurate results and an increased number of outliers. Take the 0.99-q DH-SGD in figure 51 for example, although most of the errors values are within the range $(-2^0, 2^2)$, the existence of two bigger similarity values at $(-2^6, -2^5)$ and $(-2^4, -2^3)$ drag the mean error value further away from 0. It is fair to argue the estimation error is still within the accuracy requirement range, but it is worse compared with the result in SGD, even though the 0.99-q in SGD does not even converge. Although the fluctuation has been decreased by DH-SGD (figure 50), the overall error values are also affected by only a small amount of estimations with bigger fluctuation.

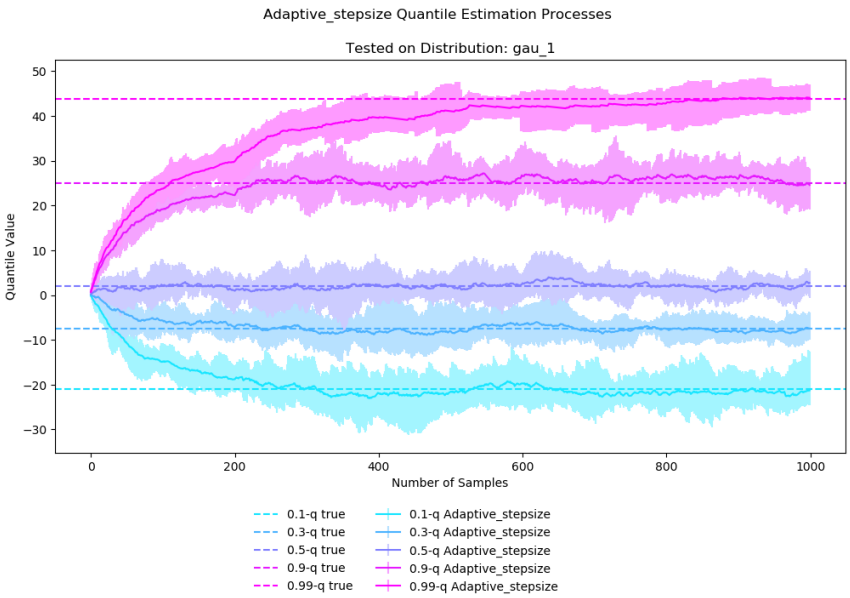

Figure 49: DH-SGD Process from *gau-1* Distribution

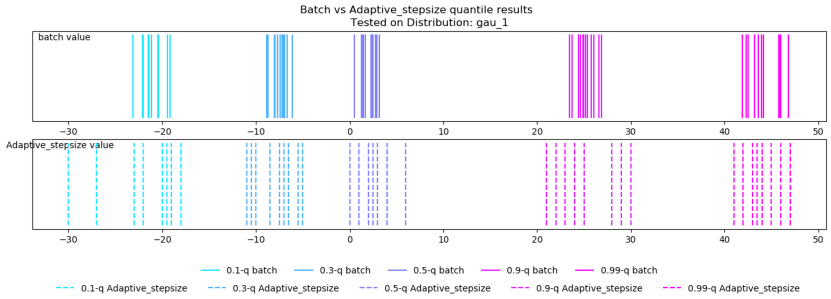

Figure 50: DH-SGD Results from *gau-1* Distribution

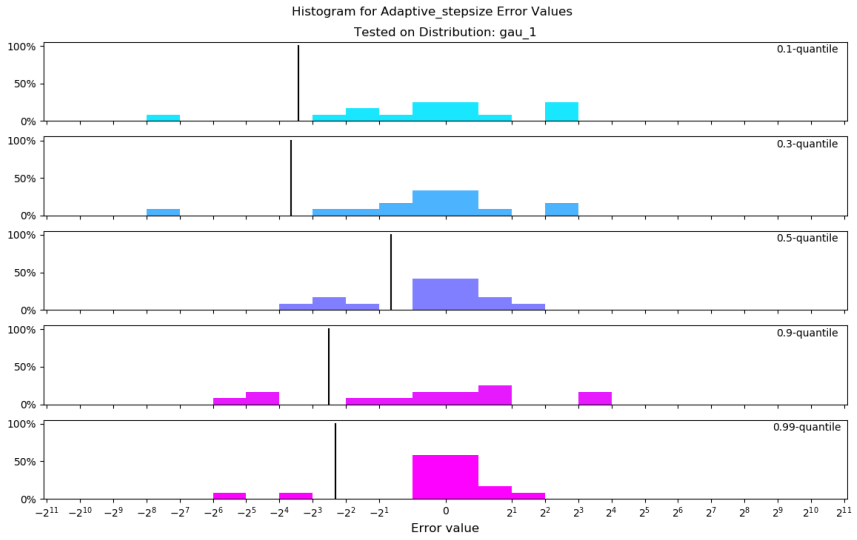

Figure 51: DH-SGD Error from *gau-1* Distribution

5.3.3 *Improvements on DH-SGD*

The small amount of estimations with fluctuation problems is a signal that the current standards for step size adaptation needs improvements. There are two potential ways for the improvements.

The first one is to change the epoch check size c, which determines how often to change the step size during the estimation. As the size and distribution of data stream is unknown, it is possible for the pre-defined parameter to be either too small or too big. If c is small, it is more likely that the c most recent updates are not a representative sample of the data, leading to the wrong change in step size. For example, if $c = 3$, it is likely that all the recent 3 updates are in the same direction even when the estimation has converged. In this case the step size is doubled when it should have been halved. On the other hand, if c is big, for example, 100000, then it takes much longer for the convergence acceleration to happen. However, we cannot know beforehand which setting is more suitable for the unknown data stream. In this situation, the choice of epoch check size is hard to be improved.

The second one is about the standards on how close is P_τ from P_τ^*. The current measurement takes the same value for all $\tau \in (0, 1)$, while a more accurate distance can be calculated from quantile probability τ and epoch check size c.

Specifically, one SGD update has only two options: increase or decrease (we ignore the option to stay the same because the probability is 0). Ideally if the quantile has already reached convergence, their probabilities are $1-\tau$ and τ. The probability of having x increase updates from the dataset of size c is now a binomial distribution problem, where the success probability $b = 1-\tau$. Generally, the possibility of having exactly x successes with possibility b from n trials is:

$$F(x; n, b) = \sum_{i=0}^{x} \binom{n}{i} b^i (1-b)^{n-i} \tag{37}$$

And the distribution of the binomial probabilities are shown in figure 52

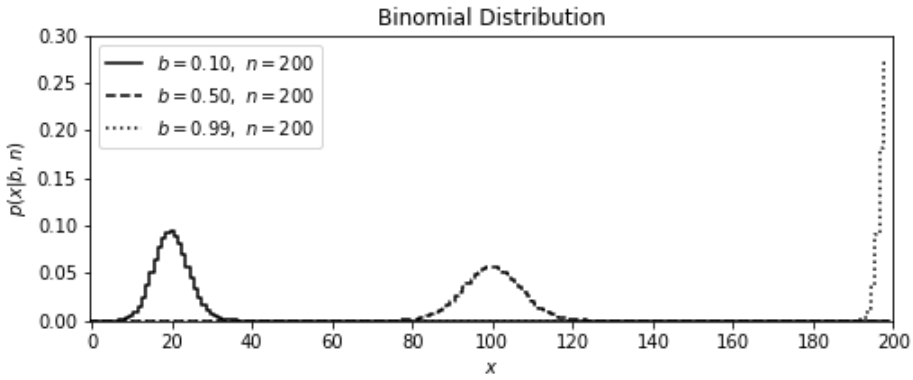

Figure 52: DH-SGD Error from *gau-1* Distribution

As shown in the plot, the distribution of the number of successful trials is different and is dependent on the success probabilities. It means the same difference $|P_\tau - P^*_\tau|$ for different τ-quantiles has a different likelihood. Using this knowledge, we can improve DH-SGD by computing the distribution of the number of increases of each quantile and derive their distance standards respectively.

5.4 CONCLUSION

To sum up, SAG and DH-SGD both improve the convergence rate of SGD. SAG improves the complexity of convergence rate in theory, but in practice might slow down the convergence when the step size scaler is smaller that the SGD default step size 1 and the convergence of SGD is already fast. The DH-SGD algorithm, on the other hand, has illustrated obvious convergence improvements on all quantiles in experiments while its convergence can not be guaranteed.

The fluctuation rate after convergence, however, is solved by neither algorithm. The step size scaler of SAG stays consistent for any updates, so the fluctuation is not improved. DH-SGD changes the step sizes regularly but does not prevent the inaccurate estimation caused by non-convergence. A potential research direction is to alleviate the fluctuation and convergence problem by a combination of SAG and DH-SGD.

6 SIMULTANEOUS MULTIPLE QUANTILE ESTIMATION

The previously introduced methods estimate only a single quantile. For applications which need multiple quantiles, these methods are not sufficient. In this chapter we introduce the problem of simultaneous multi-quantile estimation and two related methods. The structure is organised as follow:

Section 6.1 introduces the multi-quantile estimation for streaming data, followed by a discussion on the basic ideas to approach this problem. The next two sections will show how it is solved by methods focusing on different aspects.

Section 6.2 shows the *shiftQ* algorithm for simultaneous quantile estimation by implementing an idea similar to SGD.

Section 6.3 shows the P^2 algorithm that solves the problem in a different way.

6.1 THE PROBLEM AND OPPORTUNITY OF MULTI-QUANTILE ESTIMATION

In real life implementations, quantile estimation usually does not focus on a single quantile value. For example, a common request is to show the median (the 0.5-quantile) of the distribution, and at the same time show the outlier boundaries at ends of the distribution, like the 0.1- and 0.9- quantiles. It is also likely that multiple quantile estimates are required for a data distribution analysis.

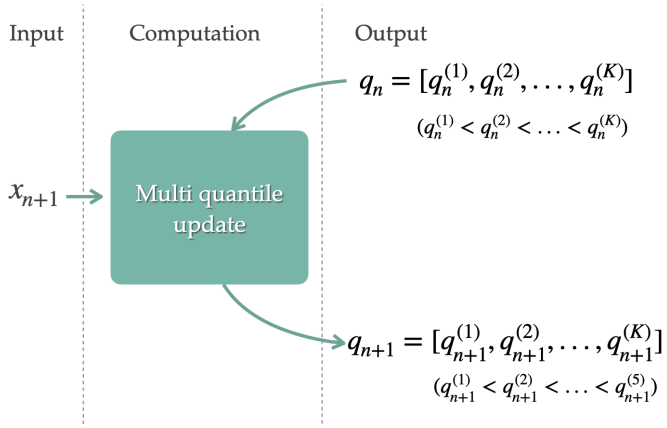

Figure 53: The update of multi-quantile estimation methods in general

A trivial solution for multi-quantile estimation is to run multiple single quantile estimation processes in parallel, such that each quantile is estimated by one process. The multi-process solution however, leads to two main problems:

1. The application software/hardware might not have sufficient parallel processing capacity for the algorithm.
2. Estimating the quantiles independently means the monotone property of quantiles is not taken into account. That is, τ_1-q $> \tau_2$-q if and only if $\tau_1 > \tau_2$.. Moreover, in section 6.1.1, we conduct an experiment on SGD on the crossing problem.

The following two multi-quantile estimation methods can simultaneously estimate multiple quantile values in one process while utilising the monotonic property. Both algorithms, *shiftQ*[15] and *extended* P^2[27] utilize the monotone property by ensuring a positive distance between one quantile and its previous quantile, i.e., $q_i - q_{i-1} > 0$. The shiftQ algorithm is described in section 6.2. It uses a method called DUMIQE, which has a very similar idea with SGD. And the next section 6.3 for the extended P^2 algorithm, which in contrast, implement a much different idea from SGD. By reproducing the two algorithms, we are looking forward to gain some insights on a SGD solution to multi-quantile estimation.

6.1.1 *The crossing problem of SGD*

In chapter 4, it is mentioned that impact of crossing on SGD appears in some experimental settings. Given the fact that the distribution of a data stream is previously unknown, the purpose of the following experiment is to briefly show the what the problem of crossing looks like and how it is affected by the settings of SGD.

Specifically, the frequency of crossing can be affected by the changes in step size. Figure 54 shows the how different step size settings of SGD affect the performance in quantile crossing. In this experiment, the data stream consists of 5000 samples from a uniform distribution $\mathcal{U}(0, 100)$, and the three SGD methods use constant step size of $5, 1$, and 0.45 respectively.

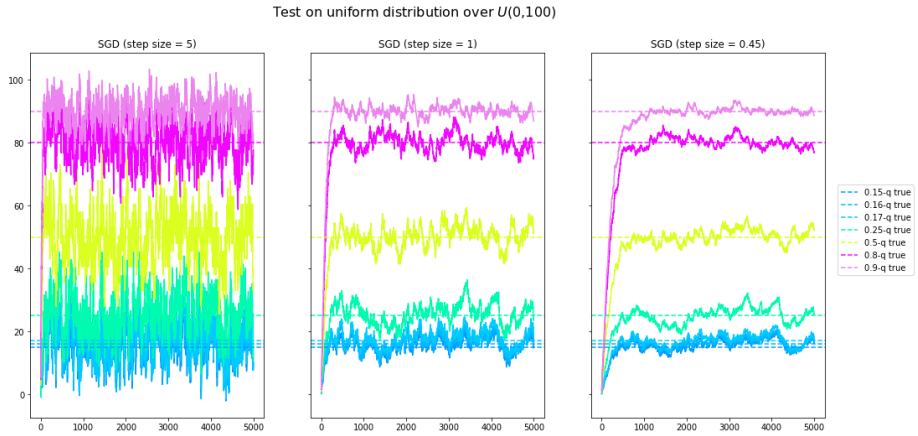

Figure 54: Three process plots comparing the estimation of quantiles 0.15-q, 0.16-q, 0.17-q, 0.25-q, 0.8-q and 0.9-q with SGD over a uniform distribution. The step size of the SGD algorithms are $5, 1$, and 0.45. In this experiment, each of them respectively has 2882, 659 and 33 epochs with crossing quantiles.

Table 1: The mean and standard deviation (std) of crossings for different step sizes with 100 experiments

Step size	Mean no. of crossings	Std Dev.	Proportion of crossings
5	2820.1	129.95	56.4%
1	676.77	177.18	13.5%
0.45	114.07	80.86	2.3%

The crossing problem in SGD has three interesting properties:

1. The number of crossings is highly sensitive to the step size. We can see from the process plots that crossings are much more common when the step size is big, which matches the data from 100 repeated experiments that 56.4% of epochs have crossings with step size = 5, while crossings were only observed in 2.3% of epochs for step size = 0.45 (table 1).

2. The crossings are likely to occur when two quantile values are close to each other relative to the SGD step size. For example, when $\alpha = 0.45$, only two pairs of quantiles (0.15, 0.16) and (0.16, 0.17) are detected to have crossings. The difference between the true quantile values of these pairs is 1, while the other adjacent quantile pairs are at least 8 away from each other. Given the step size was set to $\alpha = 0.45$ for SGD, the short distance between two adjacent quantile values are likely to be the main cause of crossings.

3. The crossings tend to occur in a slot of concentrated continuous epochs. Consider, for example, the case where $\alpha = 0.45$ in figure 54. Among the 33 crossings, 11 of them are between 0.16-q and 0.17-q from epoch 2400 to 2410. The other 22 crossings are between 0.15-q and 0.16-q continuously from epoch 3489 to 3510. This suggests that once a crossing takes place, its impact is likely to continue for the following epochs until the crossing is resolved. On the other side, when the step size is big, the crossings are distributed in a more random and even way (i.e. step size = 5 in this situation).

6.2 THE SHIFTQ METHOD

In general, shiftQ updates its quantile estimates each time a new observation is made. Each update of the algorithm consists of the update of a central quantile followed by updating each other quantile outwards from the center. The central point is updated with the *deterministic update-based multiplicative incremental quantile estimator* (DUMIQE) and the latter are updated using the concept of *shifted distributions*. In this section, we first briefly introduce the DUMIQE algorithm, followed by an explanation of shifted distributions and section 6.2.1 details how the shiftQ algorithm uses these concepts to estimate multiple quantile simultaneously.

The DUMIQE algorithm updates the quantile estimate upon the arrival of each new observation. It is a development on the work of Tierney[31], which applies a stochastic approximation on quantile estimation. The similarity between DUMIQE and SGD can be seen in the following algorithm pseudo-code:

The update of other quantiles are based on the the relationship with the central quantile. For example, if the central quantile is the median, the 0.25-quantile is then estimated as the median of distribution below the estimate of median. To apply this idea, DUMIQE uses a shifted distribution Y of the input X such that $X = Y + \delta$, where δ is the shift constant. In the following method description, we use $Q_X(q)$ and $Q_Y(q)$ to denote the estimate of q on distribution X and Y. In order to distinguish the quantile estimate on different distributions, the

Algorithm 7 DUMIQE algorithm

Input: Data Stream X, stepsize α, τ, initial quantile estimate q_0 ($q_0 > 0$)
Output: q

```
q = q_0                                   ▷ Positive initialization
for x_k in X do                           ▷ Parameter update for each input data point
    if x_k > q then
        q = q + ατ · q
    else
        q = q − α(1 − τ) · q
return q                                  ▷ q is the DUMIQE estimation result
```

DUMIQE algorithm for the τ_k-quantile on distribution X at the observation of the nth sample x_n is written as

$$\begin{aligned} Q_{X,n+1}(q_k) &\leftarrow Q_{X,n}(q_k) + \alpha\tau_k \cdot Q_{X,n}(q_k) && \text{if } Q_{X,n}(q_k) < x_n \\ Q_{X,n+1}(q_k) &\leftarrow Q_{X,n}(q_k) - \alpha(1-\tau_k) \cdot Q_{X,n}(q_k) && \text{if } Q_{X,n}(q_k) \geqslant x_n \end{aligned} \tag{38}$$

6.2.1 *Method Description*

Motivation: The difference between the two quantiles for the x_n observation is:

$$\mathtt{diff} = |Q_{X,n}(q_{k+1}) - Q_{X,n}(q_k)|$$

Note that $\mathtt{diff} > 0$ for all $x_i, i \in \{1, ..., N\}$, so different quantiles never cross each other. This property is guaranteed by the update function DUMIQE() and its restriction that the input quantile estimate must be positive.

At the arrival of observation x_n, the central quantile estimate q_c is first updated. Denote K as the number of quantiles for estimation. Then the quantile estimates smaller than q_c are updated based on the nearest quantile larger than them, yielding the sequence $\{q_{c-1}, ..., q_1\}$, and conversely the bigger quantile estimates are updated in the order of $q_{c+1}, ..., q_K$.

Updating one quantile: The difference between a quantile q_{k+1} and its neighbour q_k is calculated based on the idea of a "shifted distribution". Let X denote the original distribution of the data stream, and let the distribution Y denote a shifted version of X such that $Y = X + \delta$ for some constant δ. In this way, the quantile estimate q_{k+1} can be updated by implementation of shifting. The basic steps are:

1. Calculate the shift constant $\delta = Q_{X,n}(q_k)$
2. Get shifted observation $y_{n,k+1} = \delta - x_n$
3. Use the shifted observation to calculate the shifted quantile estimate $Q_{Y,n+1}(q_{k+1})$ with DUMIQE
4. Shift back to X: $Q_{X,n+1}(q_{k+1}) = \delta - Q_{Y,n+1}(q_{k+1})$

When updating a quantile q_{k-1} based on q_k, step 2 uses the inverse: $y_{n,k-1} = x_n - \delta$

shiftQ update: The steps of the shiftQ algorithm are:

1. **Update the central quantile:** Calculate $q_c = Q_{X,n+1}(q_c)$ by DUMIQE
2. **Update the smaller quantiles:** Starting from central quantile q_c, the estimates for $q_{c-1}, ..., q_1$ are calculated each based on the last.
3. **Update the bigger quantiles:** Similar to step 2, the larger quantiles are estimated for $q_{c+1}, ..., q_K$ in sequence.

Pseudo-code for the shiftQ algorithm is included as algorithm 8.

Algorithm 8 The shiftQ Algorithm

Input:
Dataset X (with positive data points only)
Target quantile values $[\tau_1, \tau_2, ..., \tau_K]$
Central position c, Number of quantiles K
Stepsize α_X, Stepsize α_Y
Quantile Estimate Initialization on X: $0 < Q_{X,0}(q_1) < ... < Q_{X,0}(q_K)$
Quantile Estimate Initialization on Y: $0 < Q_{Y,0}(q_1) < ... < Q_{Y,0}(q_K)$
Output: Target quantile estimates $[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}]$

Choose some $c \in 1, ..K$, usually the middle point
$q_c \leftarrow Q_{X,0}(q_c)$
for $x_n \in X$ **do**
    ▷ Update central quantile
    $q_c \leftarrow \text{DUMIQE}(q_c, x_n, \tau_c, \alpha_X)$

    for $k \in \{c-1, ..., 1\}$ **do**
        ▷ Update smaller quantiles
        $\delta \leftarrow Q_{X,n+1}(q_{k+1})$
        $y_{n,k+1} \leftarrow Q_{X,n}(q_{k+1}) - x_n$
        $Q_{Y,n+1}(q_k) \leftarrow \text{DUMIQE}(Q_{Y,n}(q_k), y_{n,k+1}, q_k, \alpha_Y)$
        $Q_{X,n+1}(q_k) \leftarrow \delta - Q_{Y,n+1}(q_{k+1})$

    for $k \in \{c+1, ..., K\}$ **do**
        ▷ Update bigger quantiles
        $\delta \leftarrow Q_{X,n+1}(q_{k-1})$
        $y_{n,k-1} \leftarrow x_n - Q_{X,n}(q_{k-1})$
        $Q_{Y,n+1}(q_k) \leftarrow \text{DUMIQE}(Q_{Y,n}(q_k), y_{n,k-1}, q_k, \alpha_Y)$
        $Q_{X,n+1}(q_k) \leftarrow \delta + Q_{Y,n+1}(q_{k-1})$

▷ Return final estimaton
$[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}] \leftarrow [Q_{X,N}(q_1), ..., Q_{X,N}(q_K)]$

6.2.2 *Experiment Results*

There are two experiments run on shiftQ algorithm. The first one aims to compare the quantile crossing behaviour of shiftQ and SGD, and the second one tests shiftQ's performance on quantile estimation.

In the first experiment, the experiment dataset is 5000 samples generated from a uniform distribution over the range $(0, 100)$. All data points are required to be

positive as the prerequisite of the shiftQ algorithm. This experiment compares two algorithms, shiftQ and SGD (step size = 0.45 because the behaviour of SGD is clearer for it), and examines the quantile crossing behaviour of each algorithm.

Figure 55: Two process plots comparing the estimation of quantiles 0.15-q, 0.16-q, 0.17-q, 0.25-q, 0.8-q and 0.9-q with shiftQ and SGD over a uniform distribution.

Figure 55 shows the estimates by shiftQ and SGD over 5000 epochs in the first experiment. In this experiment, 0 crossings occurred in the shiftQ implementation, while SGD caused 25 crossings. Over 100 repetitions of this experiments, only 3 SGD experiments had 0 crossings, while all of the shiftQ algorithms ensured 0 crossings. The empirical experimental results have illustrated the effectiveness of crossing prevention on quantile estimation.

The second experiment aims to investigate the effectiveness of shiftQ algorithm. Figures 56, 57 and 58 show process plots for the shiftQ algorithm run on the absolute value of 1000 samples from *positive gau-1*.

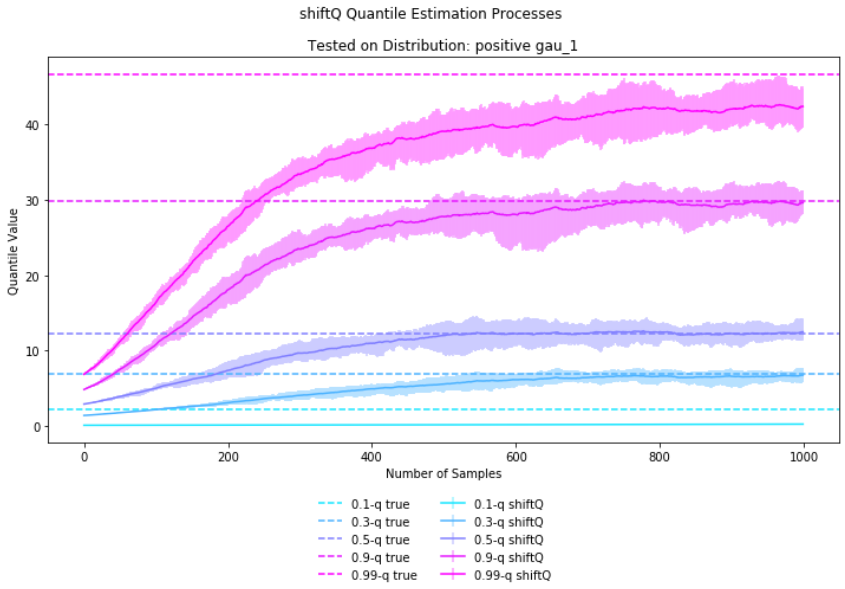

Figure 56: The shiftQ algorithm for Positive Gaussian 1 distribution (the process graph)

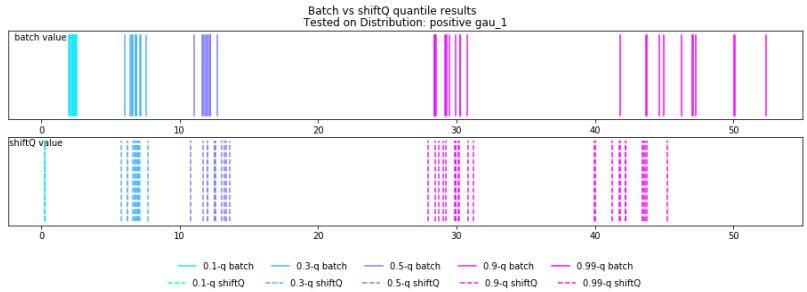

Figure 57: The shiftQ algorithm for Positive Gaussian 1 distribution (the result graph)

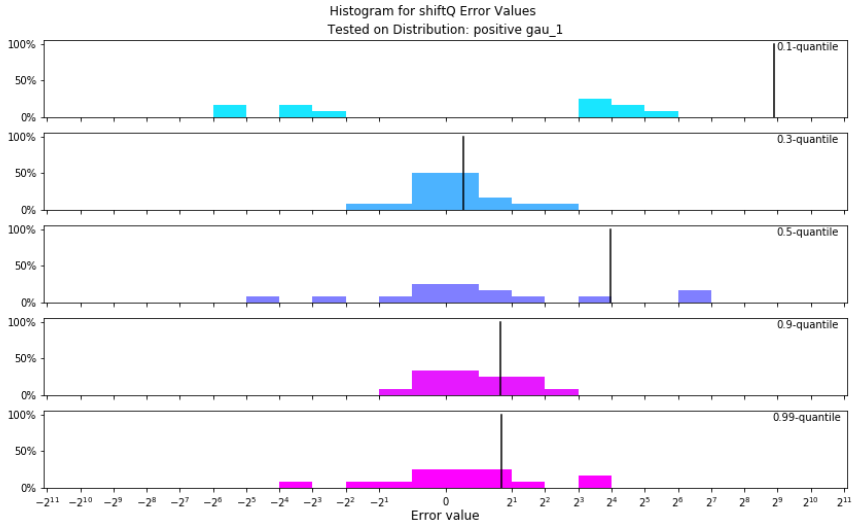

Figure 58: The shiftQ algorithm for Positive Gaussian 1 distribution (the error graph)

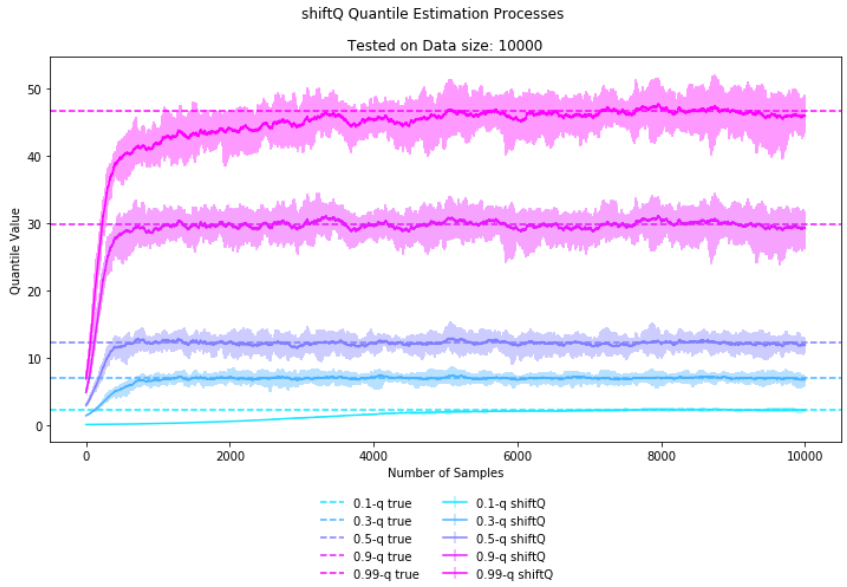

Figure 59: The shiftQ algorithm for Positive Gaussian 1 distribution (the process graph)

In the process plot in Fig 56 the quantile estimates for 0.1-q and 0.99-q have not converged even after 1000 epochs. From the comparison with the 0.5-q we can see that this big error is not caused by the initialization value of 0.1-q. Specifically, the

shiftQ 0.5-q converges even though the true quantile of it is further away from its initialization. Instead, the estimation of 0.1-q has barely changed during the 1000 epochs. The converging trend of 0.99-q is also too slow to reach the true 0.99-q quantile within 1000 epochs. The distance between the batch quantiles and shiftQ estimated quantiles in Fig 57 has shown that the quantile estimates at both ends are far away from the batch quantiles. It is shown in the Fig 58 that the shiftQ algorithm has a largely varied performance on different quantiles. For example, the 0.5 and 0.9 quantiles has a much smaller and stable error distribution around 0, while the 0.1 quantile has a wide range of error values from less than -100 to more than 50.

To test the convergence of end quantile values, a further experiment is done with a dataset of 10000 samples. Fig 59 shows that both 0.1-q and 0.99-q estimates finally converge when there are a sufficient amount of samples.

6.3 THE EXTENDED P^2 ALGORITHM

The *Extended* P^2 algorithm[27] uses exactly the same idea as the P^2 algorithm[17]. So for a better understanding, we introduce the P^2 algorithm first in 6.3.1.1, then the generation method for the extended P^2 algorithm. In section 6.3.1, the detailed algorithm for extended P^2 is provided, followed by its experiment results in section 6.3.2.

6.3.1 *Method Description*

6.3.1.1 *The P^2 Method*

The intuition of the P^2 method is the assumption that any three adjacent quantiles form a parabolic formula. Specifically, the method interprets quantile estimation as a relationship based on quantile values and their ordering positions, and applies either linear or parabolic adjustments based on the information from their neighbours.

Consider the straightforward quantile computation for $[\tau_1, ..., \tau_K]$ that sorts all observations from the dataset $X = \{x_i\}_1^N$. Let x_i denote the ith smallest value of X, then we have $x_1 < x_2 < ... < x_N$. To find the τ_i-quantile of X, we need to find the data observation at *marker position* $m_i = \tau_i(N-1)+1$. Given the marker position, we then retrieve the value of the marker, the corresponding *quantile value* $q_i = x_{m_i}$. For 3 adjacent quantiles at $\tau_{i-1}, \tau_i, \tau_{i+1}$, their quantile values can be independently computed in the same way, as shown in Figure 60.

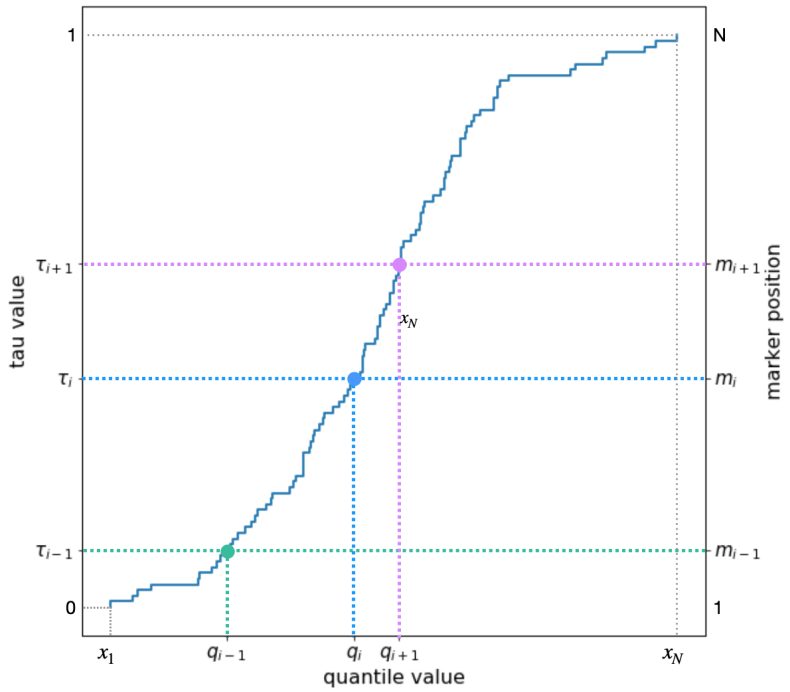

Figure 60: Relationship between τ value, marker position and quantile values correspondingly for 3 adjacent quantiles

For quantile estimation, however, storing and sorting the entire dataset is infeasible. The P^2 method records only information only about the target quantile values, and update them on the arrival of new observations. The update method,

based on different conditions, is either a *Piecewise-Parabolic* (P^2) formula, or a linear formula.

The information recorded for the ith quantile contains 3 parts: the marker position m_i, the desired marker position m'_i, and the quantile value q_i. For quantile τ_i with current observation number N, the desired marker position is $m'_i = 1 + (N-1)\tau$. This algorithm aims to keep each marker position m_i close to its desired position m'_i as new observation come in. As m_i is updated, q_i is updated to an estimate of the new value of the m_ith data point. Figure 61 demonstrates the update of m_i and q_i.

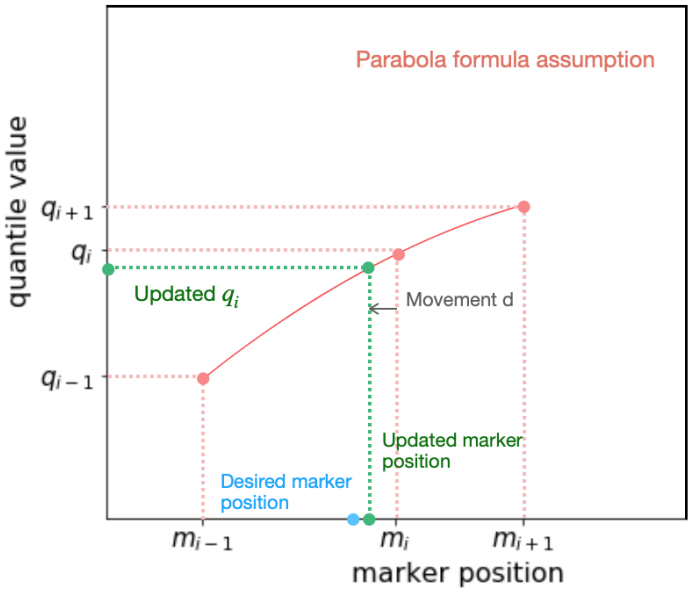

Figure 61: Quantile value update using the Piecewise-Parabolic(P^2) formula

The quantile value update is based on the Piecewise-Parabolic assumption that any three adjacent makers form a parabolic curve of the form $q_i = aq_i^2 + bq_i + c$.

If a marker is moved d positions to the right, the new quantile value is updated by the *parabolic formula*:

$$\begin{aligned} q_i \leftarrow q_i + & \frac{d}{m_{i+1} - m_{i-1}} \\ & \cdot \left[(m_i - m_{i-1} + d) \frac{(q_{i+1} - q_i)}{(m_{i+1} - m_i)} + (m_{i+1} - m_i - d) \frac{(q_i - q_{i-1})}{(m_i - m_{i-1})} \right] \\ m_i \leftarrow m_i + & d \text{ where } d = \pm 1 \end{aligned} \tag{39}$$

When applying the parabolic update would violate the monotone property, the alternative update is a *piecewise-linear formula*:

$$\begin{aligned} q_i &= q_i + d\frac{(q_{i+d} - q_i)}{m_{i+d} - m_i} \\ m_i &= m_i + d \end{aligned} \tag{40}$$

In short, the update method for τ_i-quantile takes only 2 steps:
Before starting, the P^2 algorithm extends the set of target quantiles $\tau_1 < ... < \tau_K$ with 0 and 1, giving the target quantiles $[0, \tau_1, ..., \tau_K, 1]$. When a new observation comes,

1. Update every marker position m_i to approach the desired marker position m_i'.
   a) If the quantile value is bigger than the coming observation, move the marker position by one position to the right.
   b) Adjust the marker position again if it is more than one position away from the desired marker. The move d is 1 if the marker position moves right, and -1 for moving left.
2. Update the quantile value
   a) Try applying the parabolic update to the quantile value. Note it might change the increasing order of quantiles, that is, $q_{i+1} > q_i$.
   b) If the ordering of quantile values is changed by the parabolic update, try the linear update instead.

6.3.1.2 *The Extended P^2 Method*

In P^2, the update of a quantile relies on its position relative to the neighbour markers, indicating the accuracy of neighbour marker positions is important. It sets one marker for each quantile value, that is, a total of K markers for $[\tau_1, ..., \tau_K]$. The extended P^2 algorithm improves the estimation accuracy by the introduction of "middle markers", which nearly doubles the amount of markers in P^2. This means, in the initialization part, there will be $2K + 3$ markers for

$$\tau = 0, \frac{0 + \tau_1}{2}, \tau_1, \frac{\tau_1 + \tau_2}{2}, \tau_2, ..., \tau_K, \frac{\tau_K + 1}{2}, 1$$

And the estimation update for all those markers follows the same rule as in the P^2 algorithm.

The only difference between extended P^2 and P^2 is the extension of markers at the initialization stage. At a doubly expensive computation cost, the quantile estimation reaches a higher accuracy from the extra information, as shown in the work of Raatikainen[27]. Besides the extra information brought by the extra markers, the extended P^2 algorithm also benefits from a better initialization by a larger sampling. The larger sampling in extended P^2 reduces the possibility of severely unevenly distributed initializations, which becomes important for P^2, as it is sensitive to the initialization of quantiles.

Pseudocode for the P^2 algorithm is presented in algorithm 9 followed by the extended P^2 algorithm in algorithm 10.

The extended P^2 extends the number of quantile markers of the P^2 algorithm at the initialization step, then follows P^2 in the next steps.

6.3.2 *Experiment Results*

The P^2 algorithm is tested on the Gaussian 1 distribution on 1000 samples.The process plot Fig 62 shows the convergence of different quantiles have slightly different pace, in that the 0.9 and 0.99 are very close to the true quantile value while the other 3 have already stayed stable at the 1000th epoch. Fig 63 shows the final results are close, which is evaluated in Fig 64 that error values of extended p^2 estimates has limited error values between -20 and 10 for all quantiles except for

Algorithm 9 The P^2 Algorithm

Input: Dataset X, K target quantile values $[\tau_1, \tau_2, ..., \tau_K]$
Output: Targe quantile estimates $[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}]$

o.Extend number of markers from K **to** $K' = K + 2$
▷ Add 0 and 1 before and after the target quantiles.
$[\tau'_1, \tau'_2, ..., \tau'_{K'}] = [0, \tau_1, \tau_2, ..., \tau_K, 1]$

A. Initialization
The first K' observations (sorted): $\{x_1, x_2, ..., x_{K'}\}$
for $(i = 1, ..., K')$ **do**
    Marker height: $q_i = x_i$
    Marker position: $m_i = i$
    Desired Marker position: $m'_i = ((K') - 1)\tau_i + 1$

B. For each new observation x_j**,** $j \geqslant K' + 1$**, perform the following**
switch s **do** ▷ Find the cell k such that $q_k \leqslant x_j < q_{k+1}$
    case $x_j < q_1$
        $q_1 = x_j, k = 1$
    case $q_i \leqslant x_j < q_{i+1}$
        $k = i$
    case $q_K < x_j$
        $q_K = x_j, k = K' - 1$

$m_i = m_i + 1; i = k + 1, ..., K'$ ▷ Increment markers above new observation
$m'_i = m'_i + \tau'_i; i = 1, ..., K'$ ▷ Update all the desired positions

Adjust marker heights 2 to $K' - 1$ if necessary:
for $i = 2, 3, ..., K - 1$ **do**
    $d_i = m'_i - m_i$
    if $(d_i \geqslant 1$ and $m_{i+1} - m_i > 1)$ or $(d_i \leqslant -1$ and $m_{i-1} - m_i < -1)$ **then**
        $d_i = sign(d_i)$
        $q'_i = \text{parabolic}(q_i)$ ▷ Try the P^2 update
        if $q_{i-1} < q'_i < q_{i+1}$ **then**
            $q_i = q'_i$
        else ▷ Else use linear update
            $q_i = \text{linear}(q_i)$
        $m_i = m_i + d_i$ ▷ Update marker position

C. Return quantile estimates
▷ The result is available after any number of observations
$[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}] = [q_2, q_2, ..., q_{K+1}]$

Algorithm 10 Extended P^2 Algorithm

Input: Dataset X, Demanding quantile values $[\tau_1, \tau_2, ..., \tau_K]$
Output: Demaning quantile estimates $[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}]$

1:
2: **0.Extend number of markers from** K **to** $2K + 3$
3: ▷ Evenly fill the intervals between 0, 1 and each 2 quantile values
4: $[\tau'_1, \tau'_2, ..., \tau'_{2K+3}] = [0, \frac{0+\tau_1}{2}, \tau_1, \frac{\tau_1+\tau_2}{2}, \tau_2, ..., \tau_K, \frac{\tau_K+1}{2}, 1]$
5:
6: **1. Apply P^2 with the extended initialization**
7: ▷ And returns an estimate of extended list
8: $[q'_1, ..., q'_{2K+3}] = P^2\ (X, [\tau'_1, \tau'_2, ..., \tau'_{2K+3}])$
9:
10: **2. Return quantile estimates**
11: Extract the quantile estimates for the original M quantile values
12: $[\tau_1\text{-q}, \tau_2\text{-q}, ..., \tau_K\text{-q}] = [q'_3, q'_5, ..., q'_{2K+1}]$

one outlier at -80. Overall the extended P^2 method has a really fast convergence rate.

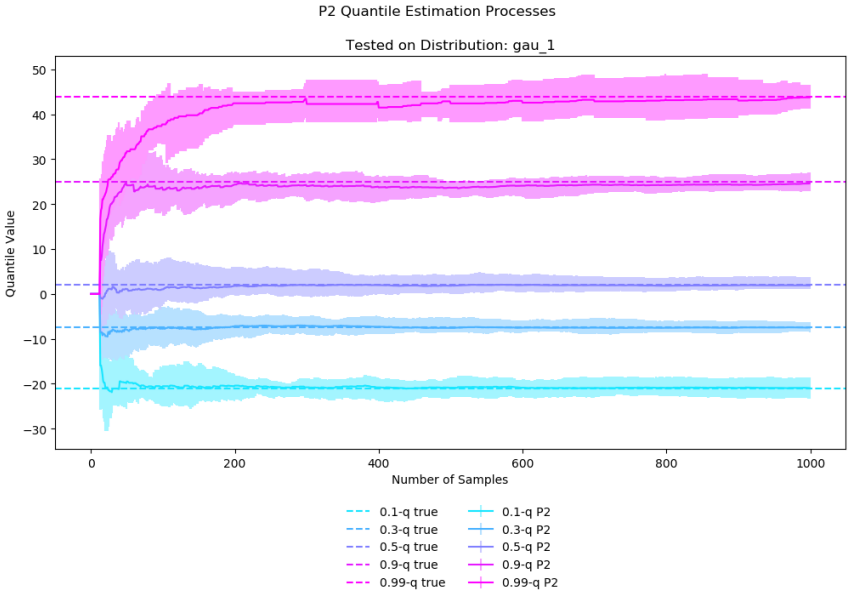

Figure 62: The extended P^2 algorithm for Gaussian 1 distribution (the process graph)

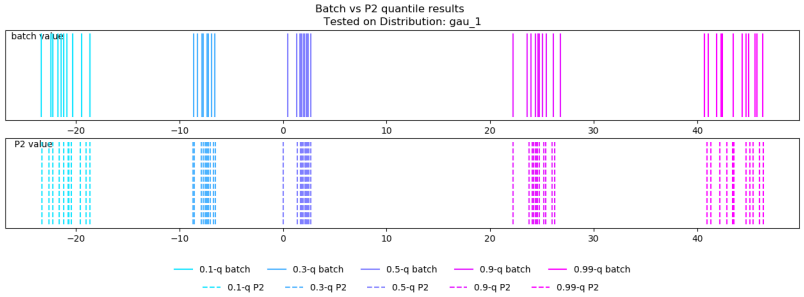

Figure 63: The extended P^2 algorithm for Gaussian 1 distribution (the result graph)

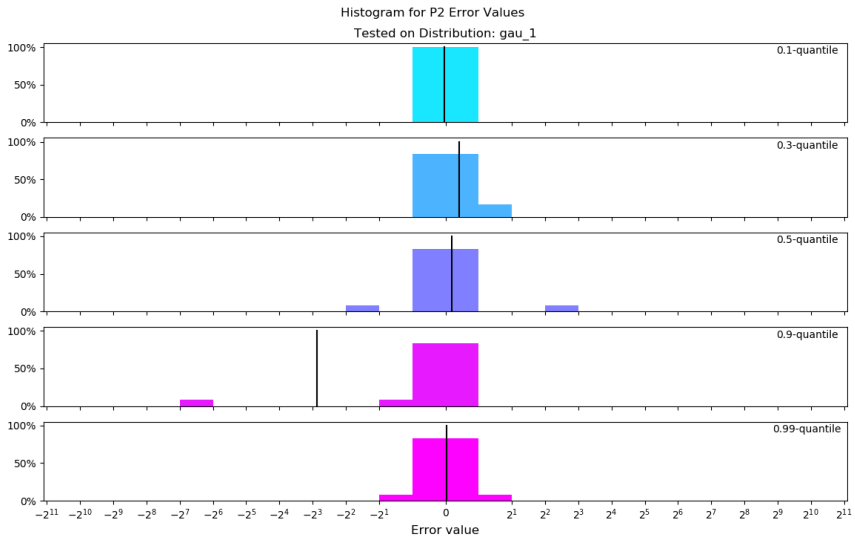

Figure 64: The extended P^2 algorithm for Gaussian 1 distribution (the error graph)

6.4 DISCUSSION AND CONCLUSION

It is clear that the extended P^2 algorithm has a better performance than shiftQ in both convergence rate and estimation accuracy. Here we list some reasoning of the performance difference, and notice that due to the fact that the two algorithms implement entirely different ideas, it is likely that some important factors are missed in this short analysis.

1. P2 has a better initialization point of quantiles, while the initialization of shiftQ is very random. The initialization of quantile estimation methods are very important, not only because the small amount of samples helps with locating of quantiles, but is also helps to find appropriate update step sizes for extended P^2.

2. The extra initialization of shiftQ makes the model tuning harder. shiftQ is sensitive to its initialization of the relative position between original distribution Q_X and shifted distribution Q_Y, which are set at some random starting points for all data distribution. In comparison, extended P^2 needs only one set of initialization points.

3. Extended P^2 uses extra information from in-between quantiles. Instead of the default K quantiles in shiftQ, extended P^2 extend it to 2K+3 quantiles. This is especially useful for multi-quantile estimation because determining the current quantile is highly dependent on the closest neighbour quantiles.

4. The mechanism of extended P^2 is to insert every new observation into an interval between corresponding markers, using information from the neighbours on both sides. The shiftQ algorithm, however, moves all quantiles based on the update from the center quantile. The updates are affected too much by only one quantile, which is likely to be amplified during the intermediate quantile updates.

5. shiftQ mskes use of less information than extended P^2 when performing updates, updating quantiles based only on the nearest neighbour towards

the center. Extended P^2 on the other hand updates quantiles based on both neighbours.

As previously mentioned at the beginning of the chapter, there are two aspects of multi-quantile estimation: reducing the crossings and using extra information from the monotone property. Both shiftQ and extended P^2 achieve the former. It is shown by repeated empirical experiments that their quantile estimates do not cross each other at any time. The latter is implemented more by extended P^2 than shiftQ, as extended P^2 uses the quantile estimate information from neighbours on both sides, while shiftQ focuses on only one side at a time. This might also partially result in the advantage of extended P^2 over shiftQ in convergence rate.

SGD is not completely appropriate for multi-quantile estimation, but by adapting these algorithms it may be. One potential future work is to implement the SGD update as a replacement of DUMIQE in shiftQ. The combination of extended P^2 and SGD is another interesting topic to investigate, as it is likely to have a better convergence rate, though whether P^2 can be adapted in this way is not clear.

7 CONCLUSION

This thesis focuses on the problem of space efficient and computationally efficient quantile estimation on data streams. Determining quantiles helps characterize a data distribution, however due to the nature of data streams, it faces challenges in both storage and computation. The calculation of batch quantiles becomes infeasible due large amount of data, coming in at a high rate. To solve this issue, algorithms of quantile estimation on data streams have been proposed which aim to keep a slow growing memory usage and low computational requirements for relatively accurate quantile estimates. In this paper, we investigate the implementation of SGD method which uses constant memory and computation time per iteration for data streams of any size.

We derive the SGD algorithm by implementing the SGD approach for quantile estimation. The proposed SGD algorithm is equivalent, both theoretically and empirically, to the Frugal-1U algorithm to some extent. Precisely, the algorithm equivalence analysis illustrates how the highly similar behaviours and results are affected by their difference in randomness. Since the evaluation results satisfy our tolerant requirements for equivalence, we conclude that the two algorithms are equal. This also implies that the SGD is a valid method for quantile estimation.

The convergence of SGD was empirically found to depend on several factors, namely data distribution, data size and SGD step size settings. Data ordering was not found to have a significant impact on convergence. Of all applications, the only factor under the control of the user is the SGD setting step size.

The newly proposed adaptation approaches of SGD step size, SAG and DH-SGD, are found to have improvement effects on convergence rate. Specifically, the SAG is theoretically proved to have a faster convergence rate while the DH-SGD convergence is tested only empirically. We discussed 2 smooth loss function approximations for SAG, but neither completely resolves the issue of convexity and continuity.

7.1 FUTURE WORK

Step size is not the only SGD setting that affects the quantile estimation performance. As is mentioned before, other SGD settings like initialization of quantile estimates are also potentially influential factors. In the current setting of SGD, the initialization of any τ-quantile ($\tau \in (0, 1)$), the initialization is always 0. The study on estimate initialization methods, for example using the first few sorted observations as starting points, is a possible direction for future work.

Another research direction is to explore more on our two current step size adaptation algorithms. Both SAG and DH-SGD are shown to have empirical acceleration effects on convergence, but they both have the unstable fluctuation problem after convergence as well as other limitation. Especially, we are concerned about the lack of convergence guarantee from DH-SGD although it provides significant improvements in some situations. It is worthwhile to consider a

new algorithm that combine the idea of both SAG and DH-SGD in step size adaptation such that the convergence is guaranteed and the step size is improved.

In addition to step size adaptation, another alternative improvement direction called simultaneous multi-quantile estimation is illustrated in chapter 6. The shiftQ and P^2 algorithms introduce different ways of keeping positive distance between each adjacent quantiles at each update iteration. It is possible that some future SGD method can implement their ideas to guarantee the monotone property of a sequence of quantile estimates.

One of the ignored issue is the use of gradient descent on the non-smooth function. Theoretically, gradient descent is not applicable for the pinball loss function which is not differentiable at 0, this application of SGD is in theory not entirely valid. However, in practice the possibility of reaching the 0 point is 0, and the setting of the gradient to zero at it makes sense. It is also shown in the experiment that the performance of the SGD algorithm works effectively in spite of the defective algorithm deduction. In theory, however, there is still work for the validness of SGD.

7.2 SUMMARY

Stochastic gradient descent is a workhorse of modern machine learning methods. Here it is used to analyse quantile estimation from data streams, and is shown equivalent to a recent method from the database systems literature. We explore the empirical performance of SGD, step size adaptation approaches and two algorithms for estimation of multiple quantiles. We hope that this connection between a machine learning approach and a streaming data problem in data analysis will provide a fruitful avenue of future research.

BIBLIOGRAPHY

[1] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. "Mergeable Summaries." In: *ACM Transactions on Database Systems (TODS)* 38.4 (Dec. 2013), 26:1–26:28. ISSN: 0362-5915. DOI: 10.1145/2500128.

[2] Arvind Arasu and Gurmeet Singh Manku. "Approximate Counts and Quantiles over Sliding Windows." en. In: *Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS '04*. Paris, France: ACM Press, 2004, p. 286. ISBN: 978-1-58113-858-0. DOI: 10.1145/1055558.1055598.

[3] Yogesh J. J Bagul. "A SMOOTH TRANSCENDENTAL APPROXIMATION TO |x|." In: *International J. of Math. Sci. & Engg. Appls. (IJMSEA)* Vol. 11.II (Aug. 2017), pp. 213–217.

[4] Yael Ben-Haim and Elad Tom-Tov. "A Streaming Parallel Decision Tree Algorithm." en. In: *Journal of Machine Learning Research* 11 (Feb. 2010), pp. 849–872.

[5] Andreas Blass, Nachum Dershowitz, and Yuri Gurevich. "When Are Two Algorithms the Same?" In: *arXiv:0811.0811 [cs]* (Nov. 2008). arXiv: 0811.0811 [cs].

[6] Chiranjeeb Buragohain and Suri Subhash. "Quantiles on Streams." In: (2009). DOI: 10.1007/978-0-387-39940-9_290.

[7] Graham Cormode and S. Muthukrishnan. "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications." en. In: *Journal of Algorithms* 55.1 (Apr. 2005), pp. 58–75. ISSN: 01966774. DOI: 10.1016/j.jalgor.2003.12.001.

[8] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. "Space- and Time-Efficient Deterministic Algorithms for Biased Quantiles over Data Streams." en. In: *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS '06*. Chicago, IL, USA: ACM Press, 2006, p. 263. ISBN: 978-1-59593-318-8. DOI: 10.1145/1142351.1142389.

[9] David Felber and Rafail Ostrovsky. "A Randomized Online Quantile Summary in O((1/ϵ)Log (1/ϵ)) Words." en. In: *Theory of Computing* 13.1 (2017), pp. 1–17. ISSN: 1557-2862. DOI: 10.4086/toc.2017.v013a014.

[10] Robert W. Floyd and Ronald L. Rivest. "Expected Time Bounds for Selection." en. In: *Communications of the ACM* 18.3 (Mar. 1975), pp. 165–172. ISSN: 00010782. DOI: 10.1145/360680.360691.

[11] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. "Chapter 40 - How to Summarize the Universe: Dynamic Maintenance of Quantiles." en. In: *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*. Ed. by Philip A. Bernstein, Yannis E. Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias. San Francisco: Morgan Kauf-

mann, Jan. 2002, pp. 454–465. ISBN: 978-1-55860-869-6. DOI: 10.1016/B978-155860869-6/50047-0.

[12] Michael B. Greenwald and Sanjeev Khanna. "Power-Conserving Computation of Order-Statistics over Sensor Networks." In: *Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '04. Paris, France: Association for Computing Machinery, June 2004, pp. 275–285. ISBN: 978-1-58113-858-0. DOI: 10.1145/1055558.1055597.

[13] Michael B. Greenwald and Sanjeev Khanna. "Quantiles and Equi-Depth Histograms over Streams." en. In: *Data Stream Management*. Ed. by Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 45–86. ISBN: 978-3-540-28607-3 978-3-540-28608-0. DOI: 10.1007/978-3-540-28608-0_3.

[14] Yuri Gurevich. "Sequential Abstract-State Machines Capture Sequential Algorithms." en. In: *ACM Transactions on Computational Logic (TOCL)* 1.1 (July 2000), pp. 77–111. ISSN: 1529-3785, 1557-945X. DOI: 10.1145/343369.343384.

[15] Hugo Lewi Hammer, Anis Yazidi, and Håvard Rue. "Joint Tracking of Multiple Quantiles Through Conditional Quantiles." In: *arXiv:1902.05428 [stat]* (Feb. 2019). arXiv: 1902.05428 [stat].

[16] Regant Y. S. Hung and Hingfung F. Ting. "An $\Omega((1/\epsilon)\text{Log}\,(1/\epsilon\,))$ Space Lower Bound for Finding ϵ-Approximate Quantiles in a Data Stream." en. In: *Frontiers in Algorithmics*. Ed. by Der-Tsai Lee, Danny Z. Chen, and Shi Ying. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 89–100. ISBN: 978-3-642-14553-7. DOI: 10.1007/978-3-642-14553-7_11.

[17] Raj Jain and Imrich Chlamtac. "The P2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations." en. In: *Communications of the ACM* 28.10 (Oct. 1985), pp. 1076–1085. ISSN: 00010782. DOI: 10.1145/4372.4378.

[18] Roger Koenker and Gilbert Bassett. "Regression Quantiles." en. In: *Econometrica* 46.1 (Jan. 1978), p. 33. ISSN: 00129682. DOI: 10.2307/1913643.

[19] John C Liechty, Dennis K J Lin, and JAMES P McDermott. "Single-Pass Low-Storage Arbitrary Quantile Estimation for Massive Datasets." en. In: (), p. 10.

[20] X. Lin, H. Lu, J. Xu, and J.X. Yu. "Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream." In: *Proceedings. 20th International Conference on Data Engineering*. Apr. 2004, pp. 362–373. DOI: 10.1109/ICDE.2004.1320011.

[21] Qiang Ma, S. Muthukrishnan, and Mark Sandler. "Frugal Streaming for Estimating Quantiles:One (or Two) Memory Suffices." en. In: *arXiv:1407.1121 [cs]* (July 2014). arXiv: 1407.1121 [cs].

[22] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. "Approximate Medians and Other Quantiles in One Pass and with Limited Memory." In: *ACM SIGMOD Record* 27.2 (June 1998), pp. 426–435. ISSN: 0163-5808. DOI: 10.1145/276305.276342.

[23] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. "Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets." In: *ACM SIGMOD Record* 28.2 (June 1999), pp. 251–262. ISSN: 0163-5808. DOI: 10.1145/304181.304204.

[24] James P. McDermott, G. Jogesh Babu, John C. Liechty, and Dennis K. J. Lin. "Data Skeletons: Simultaneous Estimation of Multiple Quantiles for Massive Streaming Datasets with Applications to Density Estimation." en. In: *Statistics and Computing* 17.4 (Sept. 2007), pp. 311–321. ISSN: 0960-3174, 1573-1375. DOI: 10.1007/s11222-007-9021-3.

[25] J. I. Munro and M. S. Paterson. "Selection and Sorting with Limited Storage." en. In: *Theoretical Computer Science* 12.3 (Nov. 1980), pp. 315–323. ISSN: 0304-3975. DOI: 10.1016/0304-3975(80)90061-4.

[26] S. Muthukrishnan. "Data Streams: Algorithms and Applications." en. In: *Foundations and Trends® in Theoretical Computer Science* 1.2 (2005), pp. 117–236. ISSN: 1551-305X, 1551-3068. DOI: 10.1561/0400000002.

[27] Kimmo E. E. Raatikainen. "Sequential Procedure for Simultaneous Estimation of Several Percentiles." In: *ACM Transactions on Modeling and Computer Simulation* 3 (1993), pp. 108–133.

[28] Carlos Ramirez, Reinaldo Sanchez, Vladik Kreinovich, and Miguel Argaez. "X2 + μ Is the Most Computationally Efficient Smooth Approximation to |x|: A Proof." en. In: (2014), p. 6.

[29] Mark Schmidt, Nicolas Le Roux, and Francis Bach. "Minimizing Finite Sums with the Stochastic Average Gradient." In: *arXiv:1309.2388 [cs, math, stat]* (May 2016). arXiv: 1309.2388 [cs, math, stat].

[30] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. "Medians and beyond: New Aggregation Techniques for Sensor Networks." In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. SenSys '04. Baltimore, MD, USA: Association for Computing Machinery, Nov. 2004, pp. 239–249. ISBN: 978-1-58113-879-5. DOI: 10.1145/1031495.1031524.

[31] Luke Tierney. "A Space-Efficient Recursive Procedure for Estimating a Quantile of an Unknown Distribution." en. In: *SIAM Journal on Scientific and Statistical Computing* 4.4 (Dec. 1983), pp. 706–711. ISSN: 0196-5204, 2168-3417. DOI: 10.1137/0904048.

[32] Sergey Voronin, Gorkem Ozkaya, and Davis Yoshida. "Convolution Based Smooth Approximations to the Absolute Value Function with Application to Non-Smooth Regularization." In: *arXiv:1408.6795 [math]* (July 2015). arXiv: 1408.6795 [math].

[33] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. "Quantiles over Data Streams: An Experimental Study." en. In: *Proceedings of the 2013 International Conference on Management of Data - SIGMOD '13*. New York, New York, USA: ACM Press, 2013, p. 737. ISBN: 978-1-4503-2037-5. DOI: 10.1145/2463676.2465312.

[34] *What Is Streaming Data? – Amazon Web Services (AWS)*. en-US. https://aws.amazon.com/streaming-data/.

[35] Anis Yazidi and Hugo Hammer. "Quantile Estimation in Dynamic and Stationary Environments Using the Theory of Stochastic Learning." en. In: *ACM SIGAPP Applied Computing Review* 16.1 (Apr. 2016), pp. 15–24. ISSN: 15596915. DOI: 10.1145/2924715.2924717.