# Assignment1. Report

## Student Information

**Name**: Anna Boronina

**Telegram**: @whoorma

**Group**: DS-18-01

**Source code**: GitHub Repository

**This report in Notion**: Notion link (easier to zoom in at images)

## Approximate quantile computation on large-scale data with MRL98 [1] and MRL99 [2]

MRL99 is built upon MRL98 with a small change: MRL98 does uniform sampling, while MRL99 does non-uniform one. Therefore, MRL98 is a deterministic algorithm and MRL99 is a randomized one [3]. I approached this assignment the same way: before implementing MRL99, I implemented MRL98.

## MRL98

It consists of three independent operations: NEW, COLLAPSE, OUTPUT. The algorithm fills and empties the buffers by alternating between NEW and COLLAPSE steps. The NEW step fills buffers with the next $k$ values from the original sequence, and the COLLAPSE step puts values from several buffers into one. These operations are performed until there are elements in the original sequence. In the end, there is just one full buffer, for which we call OUTPUT step - it calculates $\phi$ and gives the result.

A drawback of this algorithm: since we always take the next $k$ values from the sequence when creating a new buffer, the probability of newly arrived data being selected continuously goes down [3].

Manko *et. al* suggested several strategies to alternate between NEW and COLLAPSE. One of them, the most efficient at that time [1], is New Algorithm (that's the name they provided in the paper). For more information about this algorithm, refer to section 3.4 in the original paper [1]. They have proven New Algorithm to be the most efficient, so I chose it for implementation.

MRL98's implementation can be found here.

## MRL99

MRL99 is very similar to MRL98 except Manku *et. al* add a new parameter: sampling rate $r$. For MRL98 $r = 1$, but for MRL99 it changes according to how the tree height.

- it doesn't just take the next $k$ values when filling a buffer. It chooses a random index between 0 and $r$ and takes a value at that index - it's repeated $k$ times. That's why the algorithm is not deterministic anymore.

I implemented MRL99 based on the implementation of MRL98. Even though the implementation seems correct, the results are incorrect. The reason for that has not been found.

## Dataset

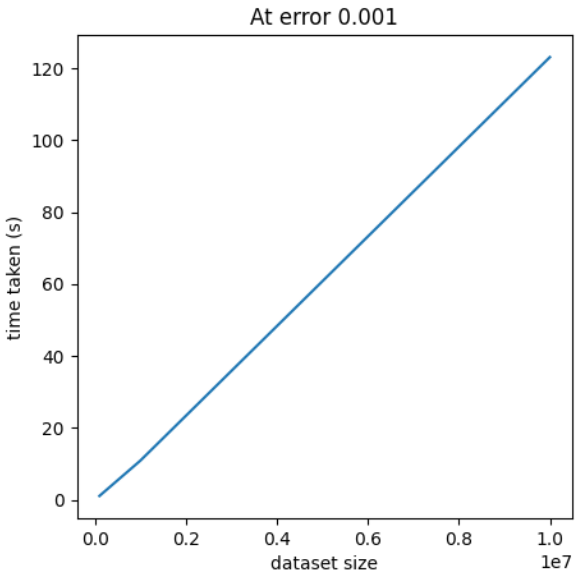
To run the algorithms, I created a few datasets: based on Gaussian, bimodal normal, Poisson distributions. All further information is provided for Gaussian distribution with $\mu = 100, \sigma = 50$ unless stated otherwise.

## Time Spent

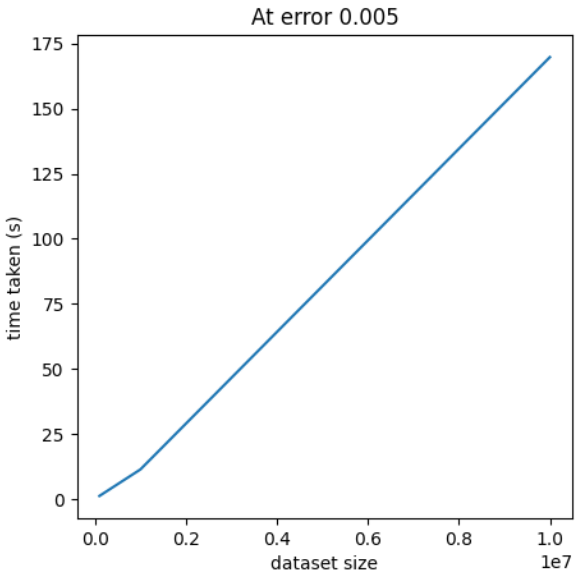
Manku *et. al* provide a table to choose the best hyperparameters for different dataset sizes and errors. The greater the dataset and the smaller the error, the more time the algorithm will take.

For error $\epsilon <= 0.005$ I considered only dataset size $N <= 10^7$ because my machine couldn't handle a higher $N$.

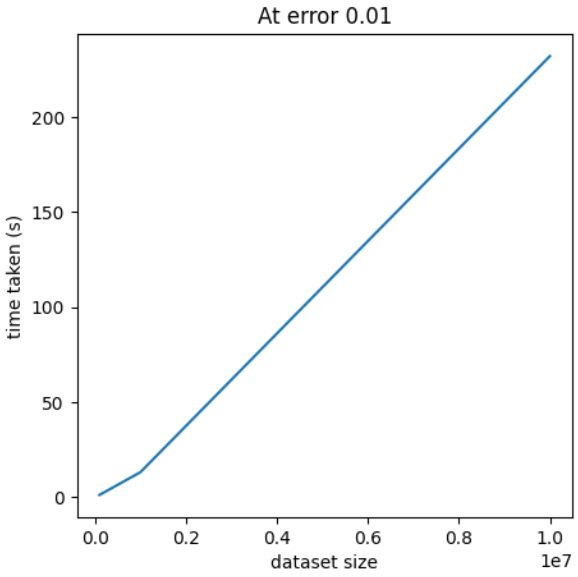
The dependencies between error rate $\epsilon$, $N$ , and time consumption are shown below.



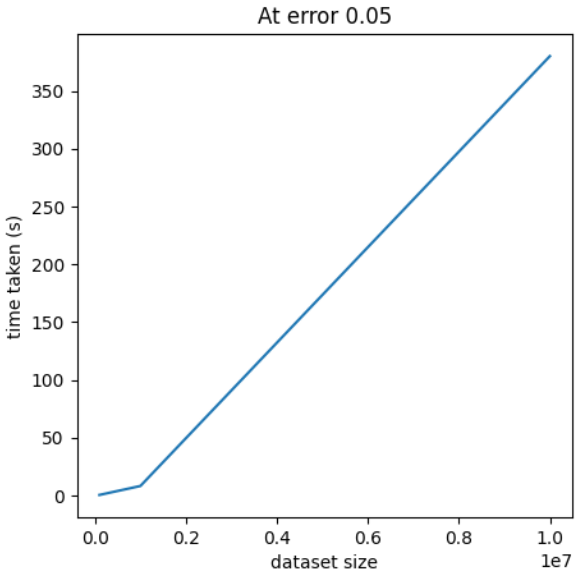

time spent depending on the dataset size for error rate 0.001




time spent depending on the dataset size for error rate 0.005




time spent depending on the dataset size for error rate 0.01




time spent depending on the dataset size for error rate 0.05

# Memory Spent

What the authors claim [1]:

New Algorithm

| | Number of buffers b | | | | | Size of buffer k | | | | | Total memory bk | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$, N | $10^5$ | $10^6$ | $10^7$ | $10^8$ | $10^9$ | $10^5$ | $10^6$ | $10^7$ | $10^8$ | $10^9$ | $10^5$ | $10^6$ | $10^7$ | $10^8$ | $10^9$ |
| 0.100 | 5 | 7 | 10 | 15 | 12 | 55 | 54 | 60 | 51 | 77 | 0.3 K | 0.4 K | 0.6 K | 0.8 K | 0.9 K |
| 0.050 | 6 | 6 | 8 | 7 | 8 | 78 | 117 | 129 | 211 | 235 | 0.5 K | 0.7 K | 1.0 K | 1.5 K | 1.9 K |
| 0.010 | 7 | 12 | 9 | 10 | 10 | 217 | 229 | 412 | 596 | 765 | 1.5 K | 2.7 K | 3.7 K | 6.0 K | 7.7 K |
| 0.005 | 3 | 8 | 8 | 8 | 7 | 953 | 583 | 875 | 1290 | 2106 | 2.9 K | 4.7 K | 7.0 K | 10.3 K | 14.7 K |
| 0.001 | 3 | 5 | 5 | 9 | 10 | 2778 | 3031 | 5495 | 4114 | 5954 | 8.3 K | 15.2 K | 27.5 K | 37.0 K | 59.5 K |

My results (I used a memory-profiler package).

| error rate | dataset size | number of buffers | number of elements per buffer | memory used |
|---|---|---|---|---|
| 0.001 | 10^5 | 3 | 2778 | 129.184 MiB |
| 0.001 | 10^6 | 5 | 3031 | 178.562 MiB |
| 0.001 | 10^7 | 5 | 5495 | 518.871 MiB |
| 0.01 | 10^5 | 7 | 217 | 145.098 MiB |
| 0.01 | 10^6 | 12 | 229 | 178.242 MiB |
| 0.01 | 10^7 | 9 | 412 | 524.453 MiB |
| 0.1 | 10^5 | 5 | 55 | 152.051 MiB |
| 0.1 | 10^6 | 7 | 54 | 174.625 MiB |
| 0.1 | 10^7 | 10 | 60 | 530.555 MiB |

While checking memory consumption, I noticed that in most cases all memory is allocated during the creation of the NewAlgorithm instance. It makes sense because according to the New Algorithm idea, all the buffers are reused and no new buffers are created.

```
  Mem usage    Increment  Occurrences   Line Contents
=============================================================
125.617 MiB  125.617 MiB           1   @profile
                                       def track_memory_usage(mrl_type: str, d, n, b, k, ph

130.352 MiB    4.734 MiB           1       data = possible_d[d](n).tolist()
130.352 MiB    0.000 MiB           1       nalg = NewAlgorithm(mrl_type, data.copy(), b, k,

130.352 MiB    0.000 MiB           1       nalg.run()
```

# Testing

I will not consider MRL99 because it doesn't work correctly. Instead, I will provide a short subsection to show that the results are incorrect.

My Null Hypothesis $H_0$ was that MRL98 is as good as `np.quantile()` in terms of the resulting value with the threshold for a p-value of 0.05. To compare them, I ran both 5, 10, 20, 30 times (5, 10, 20, 30 samples) for $\phi = 0.2$, $N = 10^6$, and the smallest error rate for MRL98: $\epsilon = 0.001$. Then I used a 2-sample t-test [`scipy.stats.ttest_ind`] to compare the results. This test outputs a p-value which shows a probability of these two algorithms giving significantly different results.

So, the result is (for p-value threshold of 0.05):

| # of samples | MRL98 mean across all samples | Numpy mean across all samples | p-value | H0 |
|---|---|---|---|---|
| 2 | 57.7571 | 57.8774 | 0.50221 | rejected |
| 5 | 57.9137 | 57.9284 | 0.84733 | rejected |
| 10 | 57.9127 | 57.9183 | 0.8780 | rejected |
| 20 | 57.8872 | 57.8877 | 0.98126 | rejected |
| 30 | 57.9303 | 57.9348 | 0.81212 | rejected |

The p-value is lower for a smaller number of samples, which led me to think I needed Bonferroni correction here, but my problem is I do not understand two things:

1. MRL98 has the highest possible precision: $\sigma = 0.001$ and MRL98's results are very close to what NumPy gives. So, why p-values are so high?
2. Do I need to use Bonferroni correction here? I think no. We need to use it when we run the test multiple times, but I ran the test only once.

So, I was confused about the results, and due to my problems with understanding these two things, I couldn't make any meaningful conclusion.

The problem for such a small p-value could be that in the paper they have hyperparameters for New Algorithm and hyperparameters for New Algorithm for 99.99% confidence. I used the first one.

# References

[1] G. S. Manku, S. Rajagopalan, en B. G. Lindsay, "Approximate Medians and Other Quantiles in One Pass and with Limited Memory", *SIGMOD Rec.*, vol 27, no 2, bll 426–435, Jun 1998.

[2] G. S. Manku, S. Rajagopalan, en B. G. Lindsay, “Random sampling techniques for space efficient online computation of order statistics of large datasets”, in *SIGMOD ’99*, 1999.

[3] Z. Chen en A. Zhang, “A Survey of Approximate Quantile Computation on Large-scale Data (Technical Report)”, *CoRR*, vol abs/2004.08255, 2020.