Designing True-Random Based Differential Privacy

Fall’23 / COMP-430 Term Project

Batuhan Arat

Hüseyin Şenol

Atabarış İlbay

Tolgay Dülger

Motivation:

As privacy attacks are becoming more and more dangerous and critical, it is vital to build novel differential privacy algorithms to prevent such attacks. As the nature of randomization, it can be accepted that it is safer compared to hardcoded algorithms with specific values. However, as stated by Canonne et al [1], pseudo-random generators are threatened by brute force attacks since they are not completely random, but randomized based on the seed value provided by the user. However, Quantum Random Number Generators (QRNG) by Australian National University provides an opportunity to develop novel and reliable algorithms. Since the numbers that randomly generated by QRNG does not rely on any pre-value or user-defined inputs, but depends on the quantum fluctuations of the vacuum,which is completely random process according to quantum mechanics, we highly believe that we can improve the security of noise algorithms, resulting in more secure and trustworthy algorithms.

Dataset & Technical Approach & Methods

The dataset used in this project is Global Land Temperatures by Country. It consists of average temperatures by each country per month beginning from 1744.

We started to research and analyze the np.random.laplace function since it is a highly popular function among researchers and engineers. This function that is written in C works as a random number generator based on Laplace distribution. There are two parameters in Laplace function.

- loc: This is the median of distribution. Therefore, this is the peak of distribution.
- scale: This is the diversity parameter, in terms of how much the distribution spreads. A smaller scale results in a steeper peak while a larger one does in a flatter graph.

The pdf of Laplace Distribution:

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

Numpy generates random numbers using inverse transform sampling. The method starts with numbers that are uniformly distributed between 0,1. Then, we also need to calculate the inverse of the CDF function, which in this case Laplace's. Inverse takes a probability between 0,1 and returns a value such that the probability of the random variable being less or equal to the given probability. We already know the CDF of Laplace (as also presented in lectures):

For $x < \mu$:
$$F(x|\mu,b) = \frac{1}{2}\exp\left(\frac{x-\mu}{b}\right)$$
For $x \geq \mu$:
$$F(x|\mu,b) = 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right)$$

Therefore, we can find the inverse of it as:

For $U < 0.5$:
$$F^{-1}(U|\mu,b) = \mu + b\log(2U)$$
For $U \geq 0.5$:
$$F^{-1}(U|\mu,b) = \mu - b\log(2(1-U))$$

Parameters μ and b are loc and scale in Numpy’s function. The major reason why this method is highly used is that it can be applied to any probability distribution as long as CDF can be inverted.

We implemented the Laplace noise function manually. As stated, we first create a uniformly distributed number, then execute the inverse CDF on that number. We create as many numbers as the number of data points. The function also has parameters as loc and scale. All code is written in Python.

We then created our custom, true-random based generator along with the noise function. We connect the ANU QRNG API to Python, and send requests. Since API returns integers between 0-255, we scale them by dividing each value by 255.

To noise the data, we first generate as many random numbers as the number of data points. Then, both noises (Pseudorandom and True-Random) are spreaded among the data that is queried by the user as passing ‘Country’ and ‘Year’. Then, histogram of the related data serves in three different ways. The real data, data noised by Laplace, and data noised by Quantum.

An error function is defined to calculate and quantitatively analyze the efficiency of the noise algorithms. We decided to use the Wasserstein Distance, which is used to calculate the distance between probability distributions on a metric M. Original-Quantum, Original-Laplace and Quantum-Laplace distances are calculated with the error function and returned as a value.

Also, budget design is implemented in the project. Since a user system is not implemented, we have defined a system-wide budget which is shared by all the clients. For development purposes, the system resets the budget whenever it is out. The system takes epsilon input from a slider in the user interface, passes it as used budget, then reduces the total budget accordingly. We started out with a small epsilon budget but considering some queries like min, max are expensive since they are very noisy for low epsilons, we raised the initial budget to about 1000 epsilons. We did not implement the slider as a percentage because getting results with the same epsilon is harder that way.

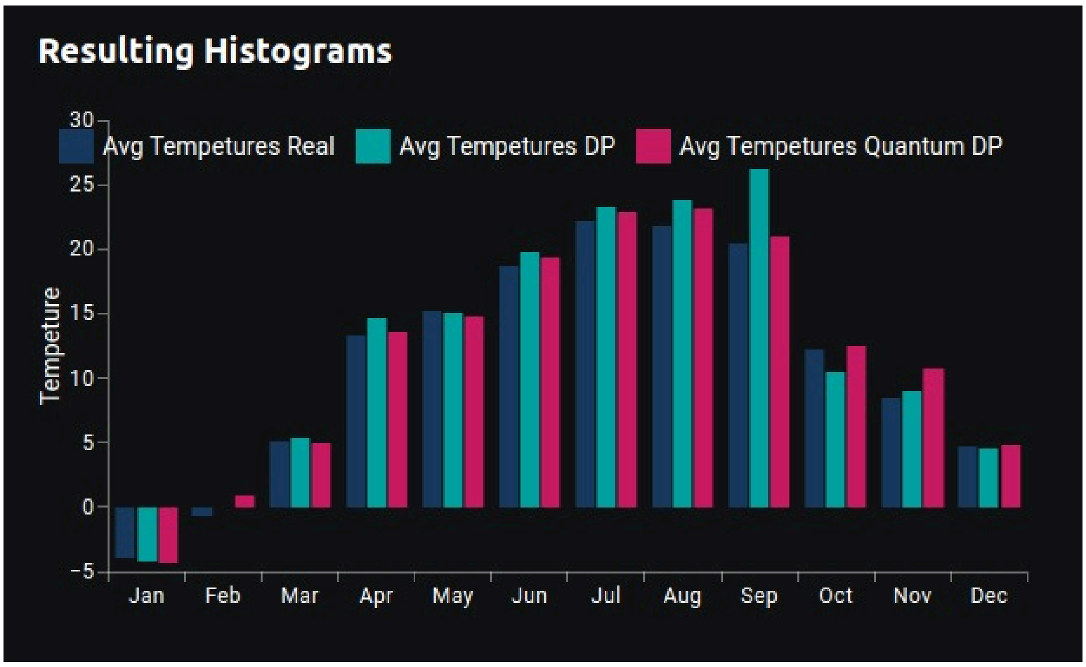


Figure 1. Example query result

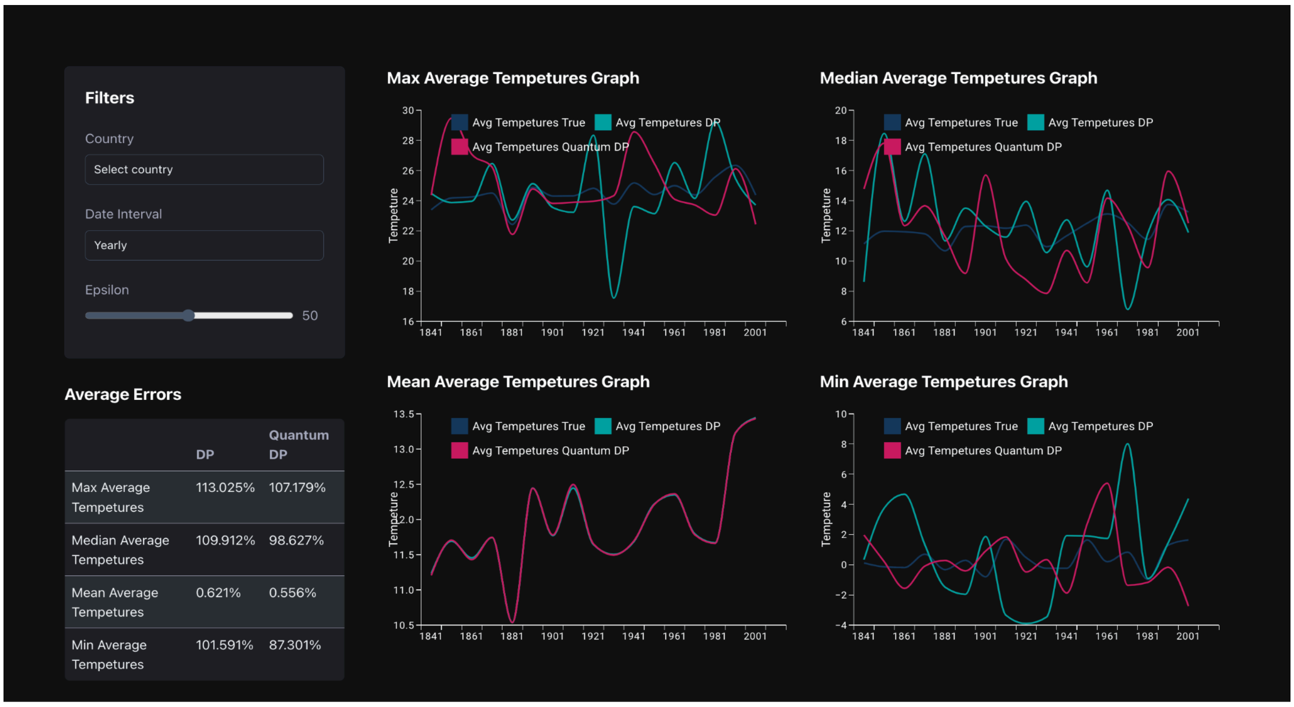
Web App

Data obtained from Pseudorandom and True-Random algorithms are not understandable and hard to comprehend without a visualization for an average person. Therefore we had to come up with a web application whose purpose is to visualize the results of the experiment and also give users a playground to experiment with different queries by changing country and year variables. For the development of the web app, we also needed a Rest API to communicate between the main logic and the front-end. For the tech stack, we have chosen to use ReactJs, a front-end JavaScript library for building web applications, with Typescript for the front-end web application and use fastAPI, a modern web framework for building APIs with Python, for the Rest API. You can access frontend project github repository via https://github.com/dulgertolgay/comp430-frontend.

The user interface consists of three parts:

- **Filters Section:** The filters allow users to filter the data by country and year. Currently, no country is selected, and the year is set to 2010.
- **Error Rates Table:** The table displays the error rates of Pseudorandom and True-Random algorithms and gives a quick numerical assessment of their accuracy.

- **Resulting Histograms:** The histograms show the average temperatures across the months of the year, from January to December, and compare three different types of average temperature data: real data without any randomness, data with Pseudorandom, and data with True-Random. The histograms visually represent how each method's output varies from the real average temperatures, helping to assess the accuracy and effectiveness of the methods.



Filters

Country: Select country

Date Interval: Yearly

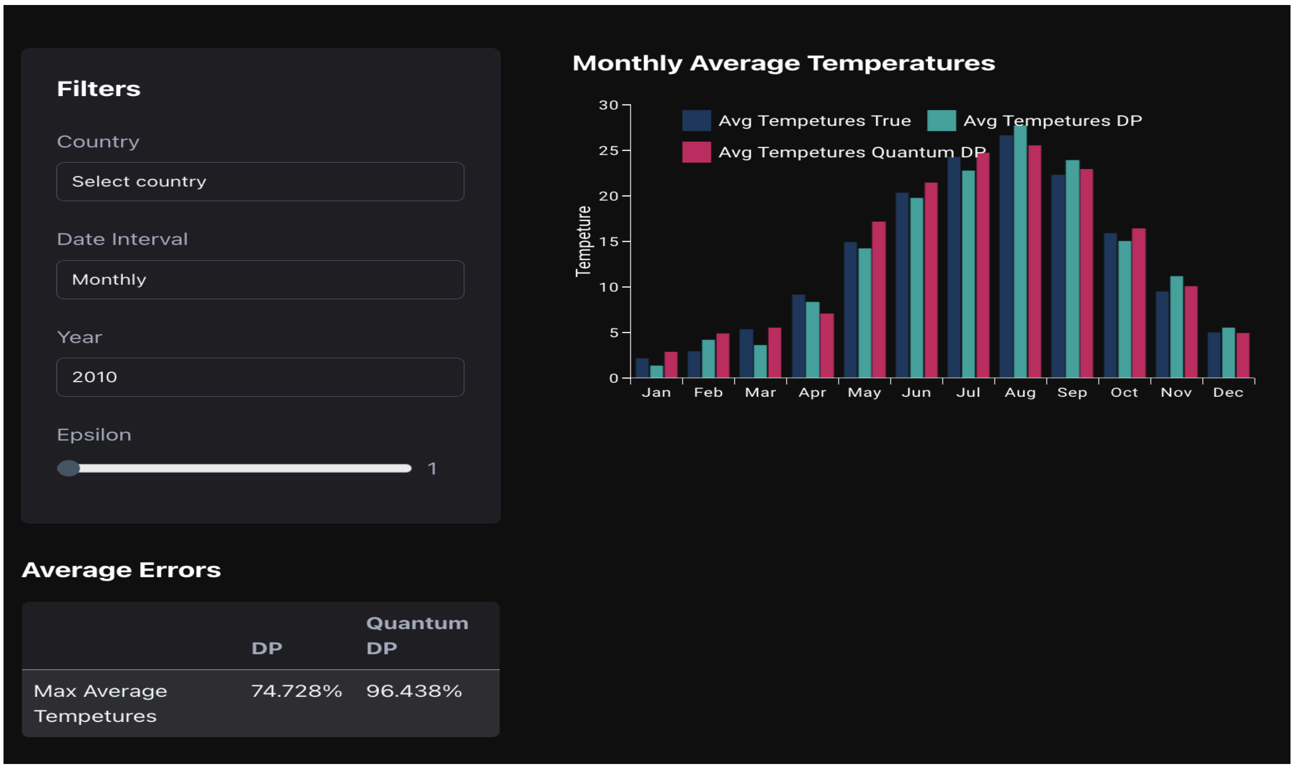
Epsilon: 50

Average Errors

	DP	Quantum DP
Max Average Tempetures	113.025%	107.179%
Median Average Tempetures	109.912%	98.627%
Mean Average Tempetures	0.621%	0.556%
Min Average Tempetures	101.591%	87.301%

Max Average Tempetures Graph
Median Average Tempetures Graph
Mean Average Tempetures Graph
Min Average Tempetures Graph
Avg Tempetures True
Avg Tempetures DP
Avg Tempetures Quantum DP
Tempeture
1841
1861
1881
1901
1921
1941
1961
1981
2001

Figure 2. Web User Interface for yearly average tempetures



Filters

Country: Select country

Date Interval: Monthly

Year: 2010

Epsilon: 1

Average Errors

	DP	Quantum DP
Max Average Tempetures	74.728%	96.438%

Figure 3. Web User Interface for monthly average tempetures

Benchmark Results

After implementing the necessary functions and algorithms, we performed experiments on the dataset with different iterations and parameters. To lower the effect of randomness, we decided to run multiple iterations. The iteration number is 19 in experiments, one of the reasons is that the API provides very little permission to generate true-random numbers while using free version, therefore we could not afford more iterations.

Firstly, we configured some parameters that are vital in the experiments. Since the maximum and minimum values in the dataset are +38 and -38, therefore we calculated sensitivity as: 76.

But, then we also modified it by dividing by 12, since we work monthly (12 months) on the dataset. Therefore, sensitivity is 6.333 in experiments. After these, we use epsilon as 3. The decision is based on the experiments that show us 3 is the best among the other possible options since it provides smoother graphs and results compared to others. Therefore, in order to increase the trustability of the experiments, we decided to move on with the smoother value 3 as our epsilon. Then, we find scale as dividing sensitivity with epsilon, resulting in 6.333 / 3 = 2.111.

Also, we calculated the parameters for yearly.

- For maximum,median,minimum → Epsilon is 40, Sensitivity is 76, therefore scale is 1.9
- For mean data → Epsilon is 1, sensitivity is 0.43, scale is 0.43
    - Sensitivity is 0.43 since 76/170 = 0.43 where there are 170 years in the dataset.

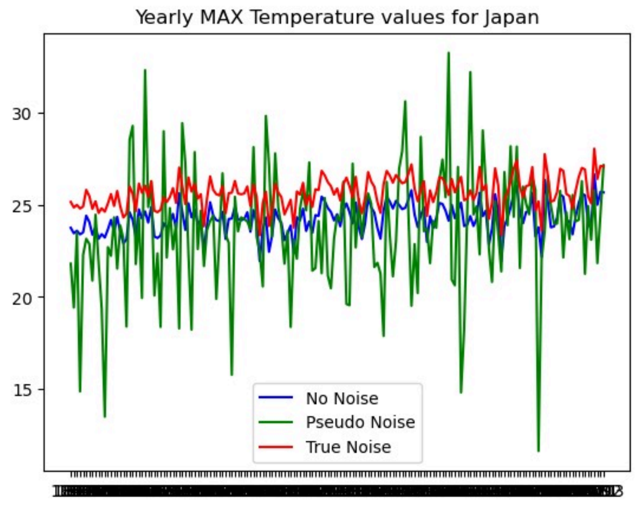


Figure 4. Yearly Max Temperature query result

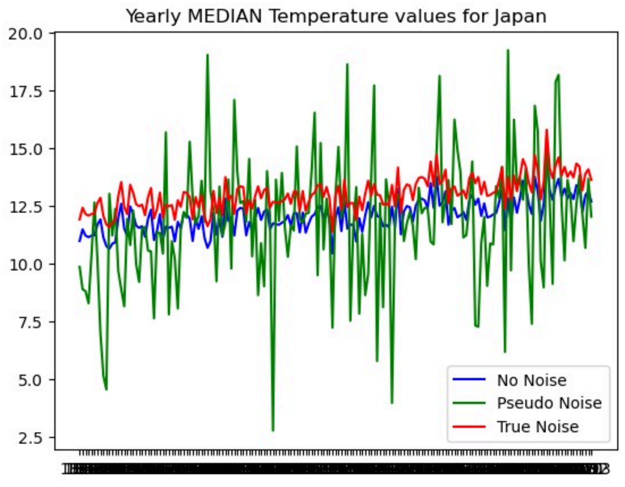


Figure 5. Yearly Median Temperature query result

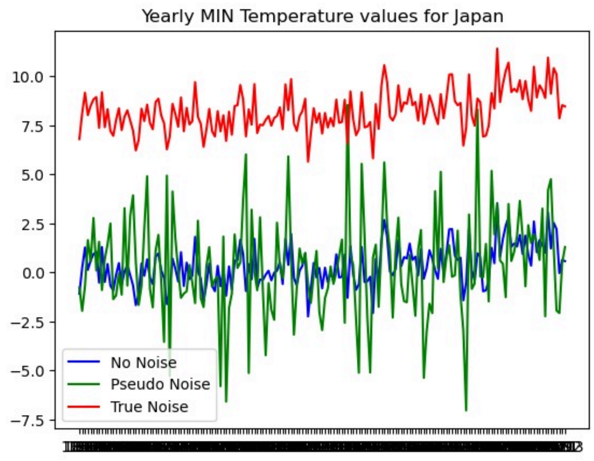


Figure 6. Yearly Min Temperature query result

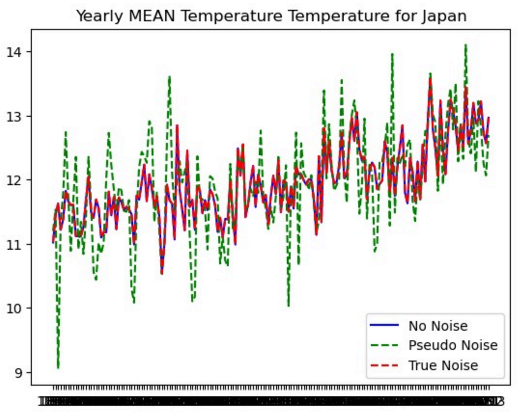


Figure 7. Yearly Mean Temperature query result

Then, for monthly, we have:

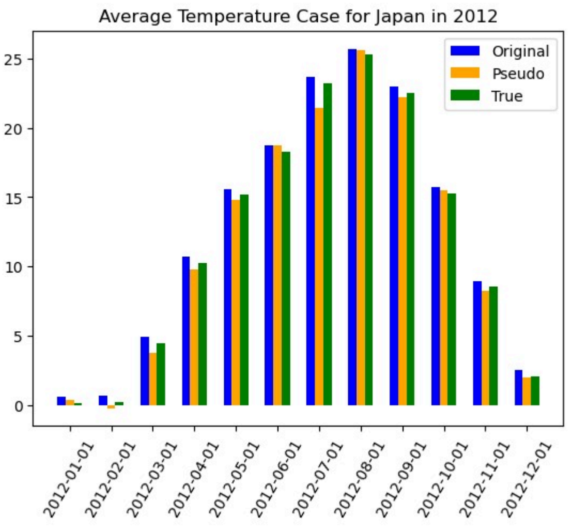


Figure 8. Monthly Mean Temperature query result

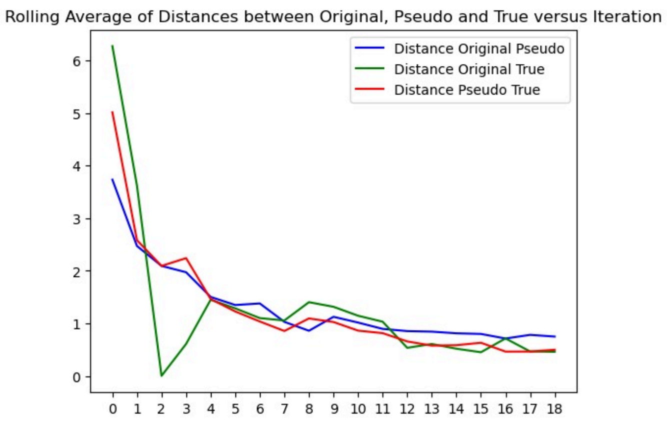


Figure 9. Average distance for each new random noise

As we start to analyze the graphs and results from the yearly data, we immediately realize that for maximum, median and minimum queries, true-random based noising is closer to real data compared to pseudo-random based noising. Especially in maximum and median queries, true-random noising clearly outperformed pseudo-random noising, which we think shows a particular use case that true-random is successful. However, true-random is significantly behind the pseudo-random in the minimum query. This may have several reasons as the dataset does not have the similar distribution in minimum values, have too many anomalies at that space of the dataset or parameters are not matchable again for the particular subset of the dataset. Therefore, we note that edge cases in the comparison must be explored deeper in a longer period as well as dataset selection is a highly crucial component of the research. The mean graph is also interesting since true-random and pseudo-random are almost identical in noise. Therefore, both noising algorithms can be used for mean queries on this dataset, since the output of them are quite similar.

For the monthly data, we analyzed the data of Japan, 2012. As seen in the first graph, true-random noise is dominant in the total 12 months, mostly closer to the real data. Again, we highlight that some number intervals (such as less and larger numbers, if we explain it simply) should be investigated separately with very large iteration numbers, since the difference is larger when the values are smaller. Also, we want to emphasize that in the iteration-error graph, it is seen that in some iterations, D(Original,True-Random) is smaller than D(Original,Pseudo-Random), where D is the Wasserstein Distance. One important difference between pseudo and true-random noise is that pseudo-random to real data distance is a smoother graph compared to true-random's. One reason as a candidate for this case is that ANU QRNG provides integers between 0-255 and we scale it by 1/255. Yet, we only generate over 255 options between 0-1, therefore in that case, it may cause anomalies resulting in peaks and valleys.

Summary

In summary, it can be said that using true-random generated numbers in differential privacy looks promising. Even though the results and error functions does not directly prove the strength and trustability of the true-random, it is visible that in some particular data points, true-random based noising was better and in some cases, Laplace noising has less errors. Therefore, one thing we want to emphasize is that QRNG can be used in a very wide range, since it does not conflict with any existing algorithm, it provides the more secure and reliable random numbers. Also, QRNG may be explored in more detail, working on some specific intervals and parameters to find the best space to be feasible, then can be used for specific research areas. Finally, the underlying algorithm of QRNG is highly impressive and safe, therefore it is almost sure that ensuring better randomization with QRNG will result in more successful noising in differential privacy.

References

[1] Canonne, C. L., et al. "Random Number Generators and Seeding for Differential Privacy." 2023. Access Article

[2] Kifer, D., Messing, S., Roth, A., Thakurta, A., & Zhang, D. (2020). Guidelines for Implementing and Auditing Differentially Private Systems. *arXiv preprint arXiv:2002.04049*, https://arxiv.org/abs/2002.04049