Lossy Compression for Quantitative Floating Point Data using Random Noise Detection

Huijten, Patrick J.

Award date:
2022

Awarding institution:
Tilburg University

Lossy Compression for Quantitative Floating Point Data using Random Noise Detection

Patrick Jozef Huijten
Data Science – Bachelor End Project
Eindhoven University of Technology
Tilburg University

Supervisor:
Erik Quaeghebeur

19-06-2022

Abstract

Our ability to store large quantities of data is steadily getting more important. This has led to the development of compression algorithms which reduce the amount of memory required to store datasets. These compression algorithms can either be lossless or lossy. In this project, a new lossy compression algorithms has been developed that allows quantitative floating point datasets to be reduced. This new method takes inspiration from other lossy compression algorithms that are centred around the size reduction of audio and image files. By classifying digits from quantitative datasets as either random noise or non-noise, insignificant digits can be removed from the dataset's contents. The amount of bits used to store the dataset's contents are then reduced as much as possible without further altering the data as to reduce the memory usage of the dataset. Two datasets were used to test this method on, one concerning metocean measurement data and the other concerning weather data. The developed algorithm reduced the memory usage of these two datasets by 45.66% and 67.53% respectively. Though the values of certain parameters have yet to be optimized, the developed algorithm has the potential to be useful in a practical setting if used in combination with other existing compression algorithms.

Table of Contents

1. Introduction

Data is rapidly turning into one of the most valuable and sought-after resources on the globe. Applications that make use of big data are becoming much more common than they once were. As a result, more and more companies are starting to realise the potential value of data and the models that can be created with it. These companies can have very different goals with the data they collect. As an example, this data can be used for the purpose of performing analyses on the current system to find out how to increase efficiency within the company. It could also be used for the purpose of predicting crucial information that would allow said company to make more informed decisions for the future. Alternatively, the data might be collected for classification or clustering purposes. Those are only a few examples of the many applications of data. Unfortunately, there is a large issue when it comes to the use of big data. As the name “Big Data” suggests, many of these algorithms require massive amount of data to be truly effective. Our ability to store massive quantities of data is thus getting more important than ever. The purpose of this project will be to find (partial) solutions to this storage problem. More specifically, to find ways of reducing the size of datasets that contain quantitative data.

1a. Lossless and Lossy Compression

When it comes to reducing the size of datasets, one needs to consider whether it is acceptable for some information within the data to be lost in the process. A data size reduction process in which no information whatsoever may be lost is called lossless compression. On the other hand, lossy compression is a process that avoids transmitting unnecessary and less-useful data, so that more storage may be saved (Marzen & De Deo, 2017). This means that some (insignificant) information is indeed lost in the process. As a result, lossy compression allows for greater data size reduction than lossless compression. This is because the methods used for lossless compression can still be used as an addition by one who intends to make use of lossy compression methods.

The figure on the right shows a simplified visualization of both lossless and lossy compression. The sizes of the boxes represent the memory usage for the dataset in question. The dataset can be completely restored to its original form for lossless compression. For lossy compression on the other hand, the restored dataset will have lost some data. After using lossy compression, it is impossible to fully restore the dataset to its original form.

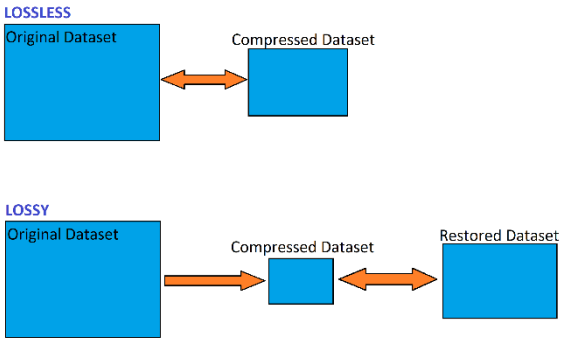

Figure 1: Simplified visualization of lossless- and lossy compression.

In the past, the primary use case for lossy compression techniques of scientific data was visualization (Cappello et al., 2019). Lossy compression methods were and still are used to store images in such a way that the image will take up far less storage while the quality of the image would only decrease slightly. Additionally, lossy compression of images causes the transmission times of said images to decrease drastically. There are many methods that were and are used for lossy compression. Coefficient prioritization, wavelet transforms and vector quantization (Goldschneider, 1997) are only a few of the many techniques used for compressing images specifically in a lossy manner. Using these methods, very large amounts of storage can be saved in settings where the extremely fine details of images are not relevant. The extent to which lossy compression can reduce the required storage space of an image greatly depends on the modality of said image. However, lossy compression generally allows the storage size of an image to be reduced by a factor of up to about 40 without causing an immediate and apparent loss in image detail when redisplayed (Allisy-Roberts et al., 2008).

The compression of images and sound had been considered to be the main and most useful purpose for lossy compression methods for a very long time. Lossy compressors designed for image processing are usually optimized considering human perception of the results. This raises an important issue for the use of lossy compression for non-image datasets. Because the optimization is based on human perception, it is very difficult to set upper bounds to the compression error that may be introduced (Cappello et al., 2019). However, in recent years there has been drastic and remarkable progress in the ability (and desire) of the scientific community to use lossy compression techniques on datasets other than only those centred around visualizations (Cappello et al., 2019). This has made it possible for many more types of datasets to be reduced in storage size using lossy compression methods.

This project will focus on lossy compression methods rather than lossless methods so that more storage may be saved. These lossy compression methods will be used in order to reduce the required storage for quantitative datasets containing floating point data in ways that will be explained in the Methods section.

1b. Implications for Supercomputers

When it comes to data size reduction, supercomputers are a very relevant factor that are only becoming more important as time passes. The processing power of these monstrous devices has grown exponentially over the last few decades. This has resulted in more and more companies taking an interest. Many of these companies are starting to make use of supercomputers to be able to run highly complex and computationally intensive simulations and prediction algorithms, often times using convolutional and deep neural networks in the process. These types of algorithms are notorious for being both computationally and memory intensive (Chen et al., 2014). It would take decades to run these specific types of neural networks on regular computers. However, supercomputers can finish running these types of algorithms in a matter of hours due to their impressive computational capabilities. When it comes to supercomputers, the reduction of data sizes is getting more and more relevant. There are many reasons for this development, but there are two reasons that stand out as the most critical ones.

The first of these reasons is closely linked to dynamic random access memory (or DRAM for short) and the development thereof in the past decade. To better understand how changes in DRAM over the years have affected supercomputers, one first needs to know what DRAM entails and why it is of crucial importance. Nearly every computer has random access memory (RAM), this refers to internal memory of the computer. This RAM stores data temporarily while the central processing unit (CPU) is occupied with the execution of other tasks. DRAM is one of the two main types of RAM, the other type being static RAM (SRAM). For supercomputers, DRAM represents a very substantial part of the total costs. And since the price of DRAM has been increasing rapidly (the average selling price tripled between 2007 and 2017 alone), the size of the system DRAM is not growing as quickly as the DRAM density (Cappello et al., 2019). This has resulted in a situation where the minimization of dataset sizes is of importance for the continuous successful development of supercomputers and the general use thereof as we move into the future.

Secondly, it is useful to consider the performance and development of sockets, which ensure that all pins on the central processing unit receive the correct signals and voltages. Additionally, the development of bandwidth between the socket of the processor and the memory should also be examined. From 2007 until 2017, the socket performance has gone up quite substantially, namely between 50% and 60% per year. This is due to the emergence and development of the multi-many-core design (Vajda, 2011). The bandwidth on the other hand, has only increased between 20% and 25% per year (McCalpin, 2016). Since these two factors have not been increasing at a similar tempo, it has created a situation in which there is an expanding gap between the processing system and the memory system. The cache hierarchy, a memory architecture that makes use of a hierarchy of memory stores based on varying access speeds to cache data, can sometimes compensate for this gap. However, it is often times not able to compensate enough and it is often not able to compensate for all applications (Cappello et al. 2019). Therefore, reducing the storage required for data is important and useful. After all, by reducing the data size, less memory bandwidth will be needed, resulting in less strain being put on the memory system. This way, more processing power of supercomputers can be put to good use regardless of the gap between processing power and the limits of the memory system.

These two factors combined make it clear to see that the reduction of dataset storage size is important for supercomputers. This displays one of the applications for data reduction algorithms. Since there is a field that needs datasets to be as small as possible to be as effective as possible, it is clear that there is a demand for such algorithms. That is why this project will work to create such an algorithm.

2. Objectives and Research Question

Before the methods and results of this project are presented, it is important to clearly and concisely set out the objectives for this project. For this section there are three key requirements that need to be satisfied for each objective. Firstly, the objective needs to be **specific**. The reason for this being that specificity can avoid ambiguity in the goal, which would make it more apparent what steps need to be taken to reach the end goal(s). Secondly, the objectives must be **achievable**. For a goal to be suitable, it needs to be grounded in reality. Objectives that could be considered to be relatively ambitious given the time constraints of this project, need to be treated with caution. The third and final requirement for the formulation of objectives is that the objective should be **relevant**. After all, for an objective to have useful real-world applications, it needs to be relevant. These three requirements for the objectives are derived from the S.M.A.R.T. method of formulating objectives (Werle Lee, 2013).

2a. Primary Goal and Significance Definition

The primary end-goal of this project is to find ways to reduce the size of quantitative datasets which store floating point values as much as possible within the given timeframe. Additionally, it is important that no significant information is lost in this process. This objective is the primary goal for this project since the need to reduce the required memory for datasets is what made this project emerge in the first place. To achieve this goal, lossy compression will be used. Since the data stored in the relevant datasets will consist of floating point values which contain many digits, it will be possible to "cut off" some of these digits so that fewer digits will be needed for each data point. This way, digits which do not add any useful information to the data point will be removed. Using this methods, less storage will likely be needed for data which contains the same information, but with fewer digits.

To better understand this goal there is one term that needs to be explored in more detail. As stated, the goal of this project will be to reduce data size without removing significant information from said data. However, in order to be able to realize this goal it is of importance to define what is meant by significant in this objective. Without being able to judge whether or not a value changes significantly, this task is not possible. Additionally, digits that are significant for one type of industry, may very well be insignificant to another industry. As an example one could consider a dataset regarding the amount of energy one could obtain from certain types of fuel. An engineer working for NASA would likely require reasonably many digits to conduct their research. Gas station owners who are interested in such a dataset on the other hand would likely not need nearly as many digits seeing as they would likely not intend to perform precise calculations with the data.

Although the significance of data often depends on the relevant field and purpose for which the data is used, there is one type of data that is insignificant regardless of the type of industry: noise. Noise is meaningless data that does not add any information to a dataset. This is because noise is the result of randomness and because noise is always assumed to not

be present in the “real” values that the data tries to represents. For example, one could think of a machine which measures a certain variable and stores the numerical results. If such a machine is able to make accurate measurements to up to four decimals, yet the measurements are stored with 6 decimals, then the final two digits will most likely be random. This means that those digits add no additional information and can be considered as noise. It can often be difficult to judge whether data is noise or actually adds potentially useful information. Finding and removing noise from data will be the main way of reducing data sizes in this project. Such a method would be useful regardless of the industry in which the data is used since, which would make such a method universal.

2b. Storage and Runtime Analyses

Another objective of this project will be to perform analyses on the effects of the amount of digits for quantitative data. More specifically, the purpose of these analyses will be to test how the runtime of basic algorithms and the amount of required storage of datasets is affected by the amount of digits of the relevant data. Since much time will be spent on reducing the amount of stored digits of data, it is important to understand how these digits relate to the data’s storage and runtimes. After all, the main end goal of this project is to reduce data sizes, reducing the amount of digits that are stored in datasets is simply a means to achieve this goal.

To perform these analyses, an arbitrary quantitative dataset will be generated since the exact values in the dataset are irrelevant in this case. This dataset will consist of many floating point values with a large amount of decimals. Then, a copy of this dataset will be created which “cuts off” the majority of these digits. Then, the amount of storage needed to save these datasets can be compared to find out what impact the amount of decimals makes on the memory usage. Additionally, some basic algorithms can be performed on both datasets and the total runtimes can be compared to find out whether fewer decimals reduces the runtimes of certain algorithms. Note that within this project, the term “digit” refers to a value from 0 to 9 rather than a binary value, which is usually used for floating point values.

Lastly, these analyses will test what impact the variable type used to store the data makes. As stated, the datasets in question will contain floating point values. However, there are many types of floats that can be used. The difference between these types of floats is the amount of bits that are used to store data. A bit is the smallest unit when it comes to computer memory, a simple binary digit. It is most common to use either 64 or 32 bits to store float values, but these are not the only options. In these analyses the impact of the amount of bits on the runtimes and required storage will be tested.

2c. F Tests

In order to remove insignificant digits from float values, one would need to find out for each decimal digit whether it can be removed without causing significant difference in the relevant value. This is where F tests may be of use. F tests are usually used to find out whether the variances of two separate normal distributions are statistically significantly different. However, F test also make it possible to find out whether there is a significant joint

difference in means for random variables (Moreira, Mexia, Minder, 2013). This means that F tests check whether a certain number of variables are likely to be all equal to each other. A goal of this project will be to find out whether F tests can be used to find out whether float values change in a statistically significant way if certain digits are removed.

This approach could certainly be of use if experimentation leads to the identification of valuable and achievable opportunities. However, since this is a very experimental and unconventional way of using F tests, this objective will not be treated as a main goal of this project. Instead, this method will be considered to be an auxiliary objective.

2d. Research Question

For this project to have a clear direction, it must have a clear, concise and relevant research question to guide it along. As discussed in section 2a, the main intended method of achieving this project's end goal of reducing data sizes will be to detect and remove digits that are nothing but noise from float values. To find out whether or not certain digits can be considered to be noise, one would need to be able to judge whether digits are fully random. After all, fully random digits can usually be identified as noise. Though it is true that there are datasets where this type of data is in fact useful and not noise, this is quite uncommon and will be elaborated upon in the discussion section of this paper. For the remainder of this project, fully random digits will be considered as random noise. Such a random noise detection analysis will be the main focus of this project and should thus be included in the research question.

Additionally, F tests could also be useful for reducing data sizes. As discussed in section 2c, this statistical method could make it possible to check whether a statistically significant change occurs when certain digits are removed. This in turn, would make it possible to categorize digits as either significant or insignificant so that insignificant digits can be removed to reduce memory usage of the dataset.

The following research question has been chosen for this project: **"How can lossy compression methods be used to reduce the memory usage of a dataset without significant information being lost?"**

This research question will be split up into two sub questions where each sub question will be centred around a possible method of reducing data sizes. The following sub questions have been chosen for this project:

1. **"How can random noise detection be used to remove insignificant digits from quantitative data in order to reduce the required memory usage for datasets?"**
2. **"How can F tests be used to remove insignificant digits from quantitative data in order to reduce the required memory usage for datasets?"**

3. Methods

Now that the objectives of this project have been described, the methods that will be used to attempt to achieve those objectives will be elaborated upon. Firstly the Law of Large Numbers will be used to develop a random-noise detection algorithm. A bit reduction algorithm centred around minimizing the number of used bits for floating point values will then be explored. Additionally, the methodology for the use of F tests in this project will also be elaborated upon. Lastly, the used datasets for this project will be described. The techniques and methodology will be documented as clearly and concisely as possible in order to allow readers to replicate the relevant experiments as well as possible.

For this project, the available literature was quite limited. A reasonable amount of research has been conducted on lossy compression methods for audio files and images. However, the amount of research that was centred around lossy compression of floating point data was small. As a result, new methods will be created in this project with the aim to fill this gap in current literature. These new methods will of course be based on knowledge gained from published literature. However, it should be noted that these techniques are experimental.

3a. Law of Large Numbers

As previously discussed, random noise detection will be of key importance for this project. As such, it is critical that potentially useful methods are identified so that research can be conducted regarding their applicability. The first method of detecting random noise will be centred around the Law of Large Numbers (LLN). The term *Law of Large Numbers* first appeared in *the History of Probability* by Poisson (1837). It was described as follows:

"Things of every kind of nature are subject to a universal law which one may well call the Law of Large Numbers. It consists in that if one observes large numbers of events of the same nature depending on causes which are constant and causes which vary irregularly,..., one finds that the proportions of occurrence are almost constant."

Thus, as the number of times an experiment is performed increases towards infinity, the average of the obtained results will converge to the expected result for each individual experiment (Seneta, 2013). The law of large numbers can be split into two categories, the Strong Law of Large Numbers (SLLN) and the Weak Law of Large Numbers (WLLN), which is also commonly referred to as Khinchin's law (Soliman, 2020). The Weak Law of Large Numbers states that the sample average converges in probability to the expected value. For any value $\varepsilon > 0$, this law can be expressed as follows:

$$\lim_{n\to\infty} P(|\bar{X}_n - \mu| \geq \varepsilon) = 0$$

Within this formula, $\bar{X}_n$ refers to the mean of the observed results. μ refers to the expected value of each individual result. And lastly, n refers to the sample size, the number of trials. This implies that for any positive specified margin ε, the probability that the difference

between the mean of the observed results and the expected value μ will be larger than ε, converges to 0.

There is a key difference between the Weak and Strong Law of Large Numbers. The Weak Law of Large Numbers states that the average of the observed values will get arbitrarily close to the expected value as the sample size grows to ∞. The Strong Law of Large Numbers on the other hand states that the average of the observed values will converge exactly to the expected value as the sample size grows to ∞ (Chen, Wu, Li, 2013). Though it is a subtle difference, it is certainly one that should be noted. For this project, the Weak Law of Large Numbers seems to be the most applicable. After all, since this theory will be used for the purpose of random noise detection and since we cannot actually increase the sample size to ∞, we are only interested in testing whether certain digits fall within a certain margin that implies that it is fully randomly distributed (this technique will be elaborated upon in section 3b).

A small-scale experiment was conducted so that the effects of the Law of Large Numbers can be visualized in an understandable and concise way. For this experiment, a certain number of random digits are chosen. The number of random digits will be referred to as n. Then, after storing these random digits, the ratio for every digit are visualized with the use of a histogram. Since the expected occurrence ratio for every digit is 10%, a red line was added to the histogram at $y = 0.10$ to display the expected results. This experiment was performed for three separate values of n: 100, 10,000 and 1,000,000. This will allow one to see what impact a growing sample size will have on the obtained results. The resulting histograms look as follows:

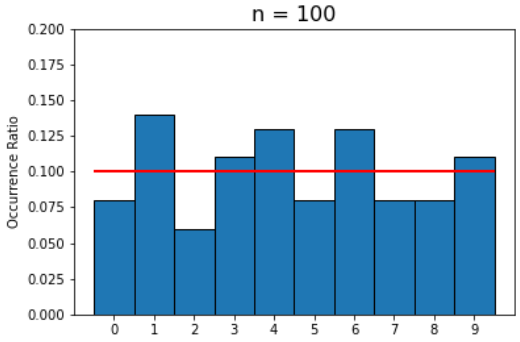

Figure 2: LLN Experiment, n = 100

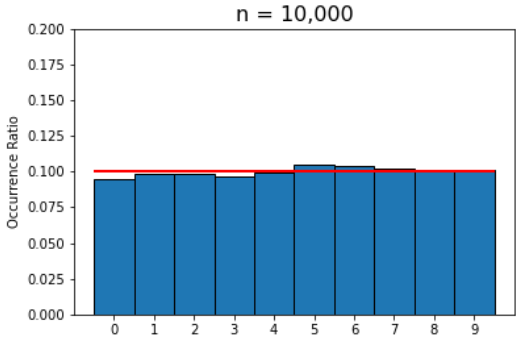

Figure 3: LLN Experiment, n = 10,000

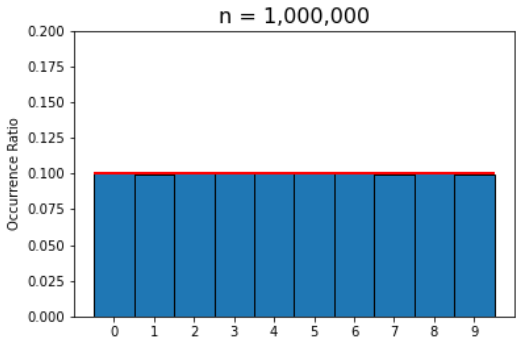

Figure 4: LLN Experiment, n = 1,000,000

As can be seen in the figures above, the ratios for the digits jointly approach the expected value of 10% as n grows. This is in line with the Law of Large Numbers and it thus confirmed and clearly showed the effects of LLN. It should be noted however, that this does not imply that the ratios will always jointly be closer to the expected value for a larger value of n. It only implies that as n increases, the odds of the ratios getting closer to the expected values will increase. There will always be a degree of randomness in the results since the sample size can never truly be equal to ∞. These findings will be useful for the development of a data reduction algorithm because as will be described in section 4e, the distribution of values of digits will be constructed similarly to the way they were constructed for this experiment (without creating a visualization). For every distribution, one could estimate how likely it is to be the result of pure randomness. This in turn, will make it possible to classify digits are either random-noise or non-noise.

To further exemplify the phenomenon described in the previous paragraph, another small experiment has been conducted. This time with the purpose to display how the error margins change as n increases. For many different values of n, n digits are randomly picked and the ratios for every digit are then computed for that specific value of n. For every digit's ratio, the difference in percentage points was then calculated between the observed ratio and the expected ratio (of 10%). The mean of these errors is calculated for every value of n to show how the mean error changes as n increases. The results of this experiment were then visualized in a simple line plot. To provide more general insights into the results, a Gaussian filter was then applied over the results so that the general trend could be observed more easily (Mathys et al., 2014). Additionally, a red line was added on the line $y = 0$ so that one can see to what degree the line approaches the expected value. The following results were obtained:

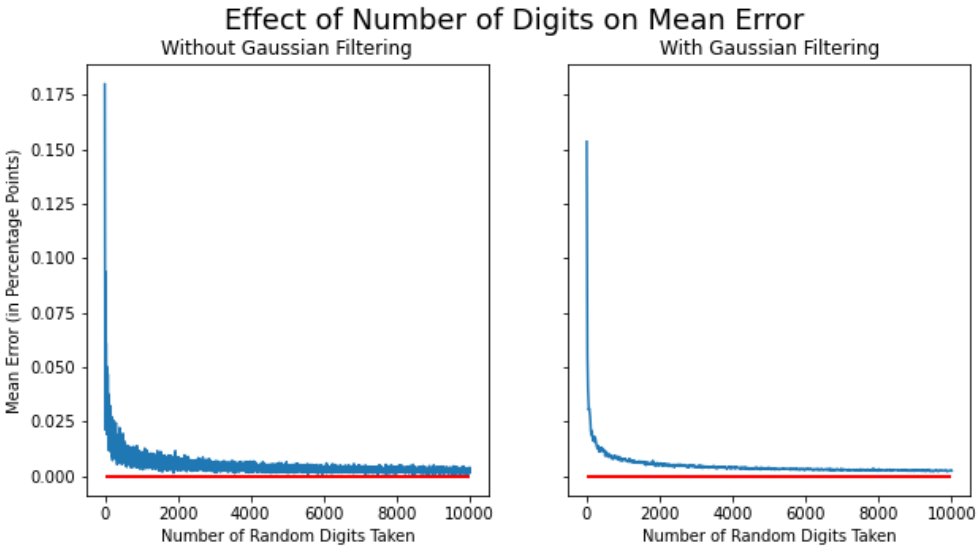

Figure 5: LLN Experiment, visualization of mean error as sample size increases

These figures clearly show that even though the mean error does not get smaller every time n grows, there is an undeniable downwards trend that moves towards the line $y = 0$. This further confirms the Law of Large Numbers and shows its effects in an easy to understand way.

3b. Random Noise Detection using LLN

Now that the Law of Large Numbers and its implications have been explored, it is time to elaborate on how this law will be used with the intent to reduce the memory requirements of datasets. To achieve the goal of this project, every decimal digit of a quantitative dataset will be analysed. The Law of Large Numbers for such a decimal digit could very well allow one to see what the expected ratio for every digit is. If these ratios are sufficiently close to the 10% point, one could assume that the decimal digit is the result of randomness. If this is the case, the decimal does not provide any useful significant information and are effectively nothing but noise (Pence, W.D., White, R.L., & Seaman, R., 2010). This would mean that the decimal can thus be removed from the dataset. It is important to note however, that this method has a strong focus on the decimal representation of floating point values rather than the more commonly used binary representation.

For the Law of Large Numbers to be applicable though, it is imperative that we have a sufficiently large sample size and that external variables remain as constant as possible. Therefore, this technique will be intended to be used for a specific types of data: dense, non-volatile time series data which results from an identical experiment or measurement being performed many times. For example, one could imagine a machine which returns measurements in the form of floating point values of a certain variable in very quick succession. If the data is collected over a sufficiently short amount of time, external factors that could affect the returned data would not be able to change significantly over the given timeframe. Additionally, it is important that the relevant data is non-volatile. If both of these conditions are met, one could divide the data into smaller parts (to keep external variables constant) and treat the data of these separate parts as though it was collected in identical conditions. This would allow one to analyse every decimal digit (starting with the last displayed digit) and find the ratios for every possible digit since the repetition of the experiment would make the observed values approach the expected values according to the Law of Large Numbers. If this analysis results in the conclusion that the digits are fully randomly distributed, one could conclude that the data of that decimal is nothing but noise which would mean that the data can safely be removed.

In order to be able to judge whether the data associated with a certain decimal place can be removed, it would need to be specified how close to true randomness the ratios of the possible digits would need to be. This parameter would greatly depend on the specific dataset that is used since it depends on how large of a sample size (value of n) can be used while keeping external variables constant. For this project's research, multiple values for this parameter will be used and the results will be compared. In the results section of this project, the impact of different values for this parameter will be shown. Then, in the discussion section, the most suitable value for this parameter will be chosen for the used

datasets and it will be discussed how one would go about choosing the most suitable value for this parameters for other datasets.

3c. Bit Reduction

In the previous section it was discussed how the amount of decimals that are stored for a quantitative time series dataset can be reduced. However, this does not yet describe how this would reduce the memory usage of the relevant dataset. By reducing the amount of stored decimals for a floating point value, the amount of bits that are used for storing said value do not change. After all, the variable type remains constant throughout this process. In order to reduce the size of the data as much as possible, one would need to check for every value in the dataset whether it can be stored using fewer bits without changing the value of the number, which is exactly what will be done in this project.

3d. F Test Methodology

In order to make the approach of using F tests for the purpose of data size reduction possible, a null-hypothesis (H_0) and an alternative hypothesis (H_a) would be required. At the current stage of the project, the following hypothesis is intended to fill the role of the null-hypothesis:

$$H_0: \alpha_1 = \beta_1, \alpha_2 = \beta_2, \dots, \alpha_k = \beta_k$$

In this formula, α_i refers to the i_{th} value of an arbitrary quantitative dataset with a total of k rows. β_i refers to the value α_i with a given number of digits removed. This test is to be performed multiple times for different amounts of digits that are to be removed from all values. This may yield useful insights into the amount of digits that can be removed without causing statistically significant changes. The advantage of this type of F tests is that it is reasonably simple to compute and it does not require the user to be careful about number of basis functions required (Moreira, E.E., Mexia J.T., Minder. C.E., 2013). The alternative hypothesis H_a would be the following:

H_a: At least one variable α_i is not equal to β_i.

Should this allow one to remove digits from floating point data, the bit reduction technique that was described in section 3c could be used to reduce the required memory space for such datasets.

3e. Datasets

Before the results of the methods that have been described will be presented, the datasets that they will be applied to must be described. There are two main datasets that will be used for this project, each with its own defining characteristics Additionally, there is a randomly generated dataset that will be used for the storage and runtime analyses.

The first of these datasets contains metocean measurement data from the Offshore Windpark Egmond aan Zee (“OWEZ”) which is located in the North Sea, 15 kilometers off the Dutch coast. This dataset was stored using 66 separate xls (Microsoft Excel) files and every row in this dataset contains measurements relating to a specific timeframe. This dataset

contains data of 48 “channels”, each channel has five associated columns, one for identifying the channel and four statistics: Minimum, Maximum, Mean and Standard Deviation. For the majority of this project, only one channel (the horizontal windspeed) will be used for the creation of data size reducing algorithms. This is because by using one channel, the data stays a lot more organized and the runtimes of algorithms will stay as low as possible while the algorithms are not yet final. Once the algorithms for reducing storage size of the data have been finalized, it will be used on the entire dataset.

The second dataset that will be used is a dataset that stores weather data from October 2012 up until November 2017. This data was collected in 7 major cities in Canada and the United States of America and was stored using a single csv (comma separated values) file. In this dataset, the following statistics are stored for specific points in time with consistent amounts of time in-between: humidity, air pressure, temperature (in °Fahrenheit), weather description, wind direction and wind speed. For this project, the data relating to temperature will be the focus. This is because the temperature data contains floating point values with quite a large number of decimals, which is ideal for the lossy compression techniques that will be used. This weather dataset can be found on Kaggle using the following url: https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data?select=temperature.csv

The two datasets described above have two main differences which will make it quite interesting to analyse the differences is how the methods of this project will affect them. The first difference is the time intervals between rows. Whereas the OWEZ dataset has time intervals of 10 minutes, the weather dataset has time intervals of one hour. If this difference leads to significant differences in results, it would be interesting to see how the results of the OWEZ dataset would change if the data was transformed so that it would also be hourly data. The second difference between the datasets is that the data in the weather dataset contains more decimals when loaded into a Pandas dataframe. The data in the weather dataset contains up to 9 decimals. The data stored in the OWEZ dataset on the other hand contains up to 6 decimals for the Mean and Standard Deviation columns and 2 decimals for the Minimum and Maximum columns. However, since Pandas dataframes are binary data structures, it is important to note that the number is changed from the binary representation to the decimal representation when displaying the data. As a result, the number of displayed decimals is often not equal to the true number of decimals that would be used to present the number in the decimal representation. It will be interesting to see whether and how this difference will affect the results for both datasets.

The third dataset is a generated dataset containing random floating point values. These random values are generated using the “random” function from the “Numpy” library. The generated values are all within the open interval (0, 10) and contain 8 decimals after being generated. These values are then stored within the dataset using 64 bits.

4. Implementation

Before the results of the described methods will be presented, the code and techniques used to implement these methods will be explained. The programming language Python (version 3.8.5) was used for all code in this project.

4a. Hypotheses

At the early stages of this project, once the methods had been thought out and analyses were ready to be performed, hypotheses about the results were constructed. Rather than being based on analyses, experiments and code, these hypotheses are based on background research and prior knowledge. Seeing whether these expectations are correct or not could yield valuable insights into the topic at hand.

The following hypotheses were constructed for this project:

1. If the amount of **digits** used to save a floating point value decreases, the runtimes of algorithms which make use of said floating point value and the amount of memory usage required for said floating point value also decrease
2. If the amount of **bits** used to save a floating point value decreases, the runtimes of algorithms which make use of said floating point value and the amount of memory usage required for said floating point value also decrease
3. Performing bit reduction after random noise detection on the used datasets will reduce the memory usage of the datasets by at least 50%

These hypotheses will be evaluated in section **5c** so that the obtained results can be used to discuss the hypotheses.

4b. Loading & Pre-processing the Data

There are two datasets that are used for this project, the OWEZ dataset and the weather dataset. Firstly, these two datasets both needed to be loaded into a Pandas dataframe. Then, the data needed to be pre-processed so that irregularities (such as outliers and missing values) can be detected and dealt with. Loading the weather dataset was a straightforward process with no unforeseen difficulties since the data had been stored as a simple CSV file. The OWEZ data on the other hand was less straightforward. However, no serious difficulties were encountered at this stage. The data was stored using 66 separate Excel files which needed to be combined to form a single dataset while making certain that the data is stored in a chronologically correct way. Once these files were loaded into a dataframe, the required features for both the OWEZ and the weather data were selected while the remaining features were removed.

For the pre-processing of the data, there are two main irregularities that needed to be detected: missing values and outliers. Although the OWEZ data did not seem to include any clear missing values, upon further inspection it became clear that there were a few extreme outliers which were likely missing values that were given an extreme, impossible value to indicate that the value was missing. The weather dataset had a total of three missing values

over a total of 45,254 rows. Since this is not at all a significant portion of the data, these missing values were removed from the dataset by removing the entire row in which the missing value occurred.

There are many possible methods for detecting and removing outliers from data. The methods used for this project was to firstly visualize the data for the relevant columns using a boxplot so that extreme outliers (substitutes for missing values) could be detected. For example, when the values of the column “StdDev” (for the chosen channel) were visualized for the OWEZ dataset, it yielded the following boxplot:

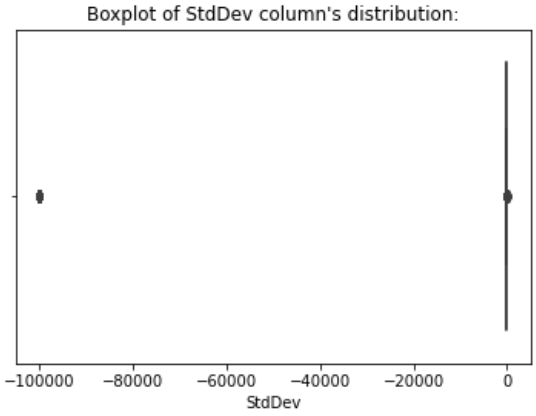

Figure 6: Extreme outliers in the “StdDev” column

Clearly, the data points around the value -100,000 are outliers which should be removed from the data. Removing these rows from the dataframe results in the following distribution:

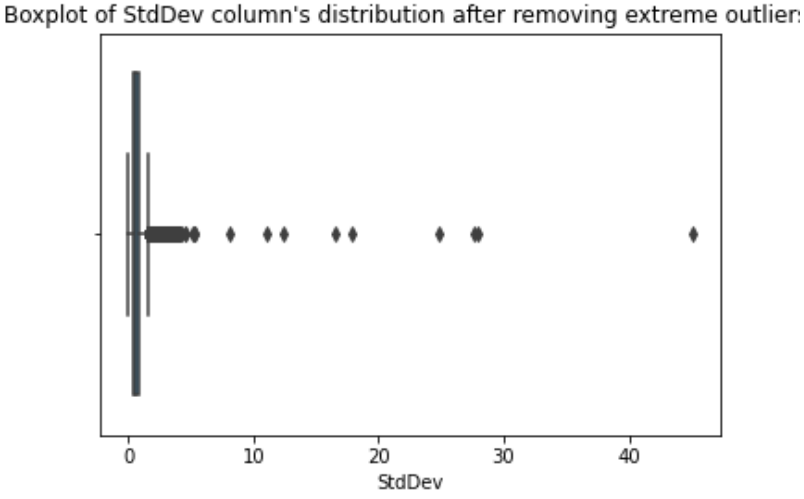

Figure 7: Remaining outliers in the “StdDev” column

Now that the extreme outliers have been removed one can see that there are still a lot of less extreme outliers that would need to be removed from the dataset. It is important to remove these less extreme outliers as well because as described in section 3b, the used data should be non-volatile for this method to work optimally. These less extreme outliers can be detected by taking the mean value μ of the column and the standard deviation σ of the column (which is not the same as the value of the standard deviation column). With these values one can construct the interval $[a, b]$ in which: $a = \mu - 2.5\sigma$
$b = \mu + 2.5\sigma$.
Values which lie outside this interval can be considered outliers (Seo, 2006) which can be removed without losing usable, significant information. It is most common to consider values as outliers if they are either more than 2 or 3 standard deviations removed from the mean value. However, for these specific datasets, removing values which are more than 2 standard deviations from the mean would have resulted in a substantial amount of data being removed. On the other hand, removing values which are more than 3 standard deviations from the mean resulted in hardly any potential outliers being removed. Therefore, in this project, outliers are defined as values which are more than 2.5 standard deviations removed from the mean value. It is crucial that the extreme outliers have already been removed before this step. After all, extreme outliers would affect the value of σ far too much for this technique to remain reliable. Once this technique had been applied on the data, 6,533 out of the 259,165 rows (2.52%) of the OWEZ dataset and 506 out of 45,250 rows (1.12%) of the weather dataset were removed as outliers. This resulted in the following distribution for the StdDev column from the OWEZ dataset:

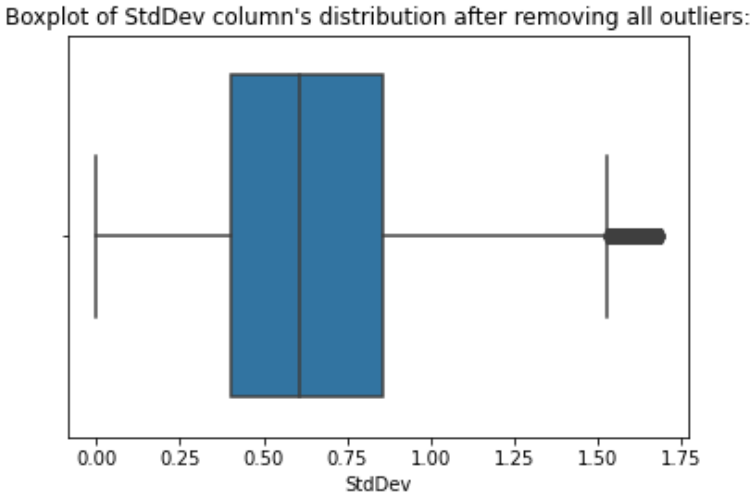

Figure 8: “StdDev” column distribution after removing all outliers

4c. Implementing Storage and Runtime Analyses

For the storage and runtime analyses of this project, there are 4 main questions that needed to be answered:

1. What impact does saving a floating point number with fewer digits have on the runtimes of simple algorithms?
2. What impact does changing a floating point number to an integer by rounding have on the runtimes of simple algorithms?
3. What impact does the amount of bits used for floating point number have on the runtimes of simple algorithms?
4. What impact does the amount of bits used for floating point number have on the required storage for the dataset?

Note that these questions are in no way replacements for the previously stated research question and its sub questions. They are simply questions that (if answered correctly) will yield insights regarding storage and runtimes that can help answer the research question.

To answer the first of these questions, the random dataset described is section 3e was generated with the use of the "Numpy" library. A second column was then created which consisted of the values of the first column rounded to 1 decimal digit. A third column was created that would consist of the values of the first column rounded to the nearest integer (the data type object of this column was also changed to integer). To test whether rounding float values or changing values to integers would have a noticeable effect on runtimes, a simple algorithms was created that would add the values of a column to each other a certain amount of times. To clearly show whether there was a significant effect, the sum of all values in the columns were added to each other 10,000 times. The results of this test clearly showed that the runtimes were not affected in any meaningful way by either of these changes. This means that simply "cutting off" digits would not have an effect on runtimes. This is likely due to the fact that the amount of bits used to store the value did not change when the value was changed to an integer.

To find out whether the amount of bits used for floats would affect runtimes, two more columns were added to the dataframe. Both of these columns contained the values of the first column. However, fewer bits were used to store said values. One column used 32 bits to store the values and the other used 16 bits. The same algorithm as previously used was then run on these columns. The results of this test were quite unexpected. The total runtime of the original (64 bit) column was 7.724 seconds. The total runtime of the 32 bit column was 1.868 seconds. The total runtime of the 16 bit column was 6.084 seconds. Thus, decreasing the amount of bits from 64 to 32 had a clear positive effect on the runtime while decreasing the amount of bits from 32 to 16 bits had a clear negative effect on the runtime. The reason for this is that consumer Intel processors like the one used for this analysis do not usually natively support FP16 arithmetic which is used for 16 bit calculations (Haidar, Tomov, Dongarra, Higham, 2018). Thus, float32 values seem to be the fastest option when it comes to computers with standard consumer processors.

The last question of this analysis was straightforward to answer. The Pandas library allows users to inspect the memory usage of each individual column of a dataframe. Unsurprisingly, the results of this function made it clear that halving the amount of bits used also halves the amount of memory that is required for the column. Since the required memory of datasets is the primary concern of this project rather than runtimes, float16 seems to be the optimal way to store data if the transition from float32 to float16 does not lead to significant non-noise data being lost.

After these analyses had been performed using the randomly generated dataset, the same analyses were performed using the OWEZ dataset. This dataset yielded results that led to the same conclusions as the analyses on the randomly generated dataset. Thus, the findings have further been confirmed.

4d. Implementing F test Data Reduction

Implementing F tests to reduce the memory usage of datasets proved to be significantly more complex than initially expected. Although writing code for performing F tests was not all that complex, there is an issue with the way this experiment was set up in the first place. As described in section 3d of this Thesis, the following null hypothesis was chosen for this experiment:

$$H_0: \alpha_1 = \beta_1, \alpha_2 = \beta_2, \dots, \alpha_k = \beta_k$$

The problem resulting from this hypothesis is that there seems to be no good way to decide on a suitable significance level. Usually choosing a suitable significance level is quite straightforward, but for this situation there are some serious difficulties. Since β_i is equal to α_i with certain digits removed, these values will always be very similar. For example, if one takes the value 1.43 for α, then the value for β could be 1.40 if only a single digit is removed. Because the number 1.43 is very close to 1.40, the null-hypothesis would almost always hold true unless the significance level is extremely small. Therefore, extremely low significance levels would be required for any result to be statistically significant. Furthermore, deciding on how low exactly this significance level would need to be seems to be impossible without "overfitting" the significance level to the specific dataset that is being used (Ying, X., 2019). This would make the analysis unusable. It may very well be possible to find a suitable significance level for this problem using advanced statistical knowledge. However, due to the lack of relevant literature regarding this approach and since the use of this method is considered as an auxiliary objective for this project, this method will be written off for this project primarily due to a lack of statistical expertise and time constraints.

4e. Implementing Random Noise Detection

Implementing random noise detection in such a way that it would allow one to remove certain digits from the stored data was the biggest challenge of this entire project. After loading the pre-processed data, the first step was to find out at what digit to start the analysis. The proposed methodology stated that the analysis would start at the last digit of the decimal representation of the float values. If this last digit were to be judged as a

random noise digit, then it would be removed and the digit in in front of that one would be analysed as well. This would continue until a non-noise digit was found. At that point, the algorithm would stop. This way, the algorithm starts at the least significant digit and work its way up to the first digit that was still behind the decimal separator. To find the starting digit for this technique, the lengths of the stored float values was saved as an additional column so that one could filter the dataset on rows where the float value has at least a certain length. This allowed for the introduction of an additional parameter which will be referred to as Q. The desired initial digit would be the last digit where at least Q% of the stored float values has a length that is large enough to have that many digits. While creating the algorithm, the value of 33.333… was assigned to Q so that the initial digit would be present in the majority of float values. The initial digit is then saved under the variable name "current_digit". The value of Q should not change for the selected datasets. Though changing the value of this parameter could result in more memory being saved, it could very well results in data being wrongfully deleted.

To judge whether or not a digit is random noise or not, an interval was constructed over the rows of the dataset. This interval would make it possible to look at certain fragment of the dataset and judge whether the digit that is currently being checked would be considered to be random noise in that specific interval. Then, the interval would move slightly and the process would start again. The length of this interval was added as a customizable parameter for this algorithm. 2 extra columns were then added to the dataframe: "random" and "not_random". These columns would contain simple integer values that would specify how many times a certain digit has been categorized as random and not random. Once the interval has moved enough to reach the end of the dataset, these extra columns would make it possible to judge whether a digit could be considered to be random or not given the results of all intervals (in which the specific row is included). Then, the algorithm would check whether there are digits at position "current_digit – 1". If so, the process would start over to check the digit in front of the previous one.

The following figure is a simplified visualisation of the process:

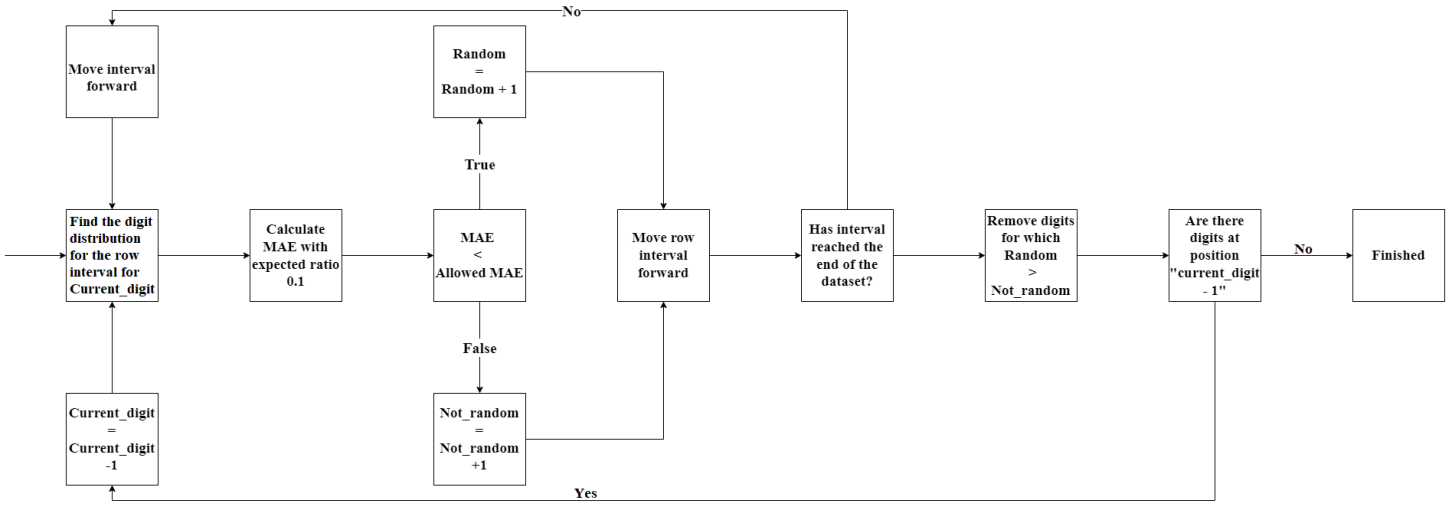

Figure 9: Simplified flowchart of Random Noise Detection Algorithm

This method resulted in a large issue however. Since the detection of a non-random digit would immediately stop the algorithm, any faulty classifications would stop the algorithms while it should have continued. Furthermore, imagine a float value for which the 4th up until the 6th digit are all random noise but the 7th digit is not noise. The algorithm as described would not remove the 7th digit and then the algorithm would stop even if the classification were to be correct. This presents problems since a non-noise digit which is preceded by many random digit could be considered as non-significant. To prevent this from problem from occurring, the algorithm was changed so that it would no longer start the analysis at the last (displayed) digit of the number. Instead, the analysis would start at the 3rd digit and move towards the last digit. Furthermore, instead of only checking whether the current digit is random or not, the algorithm would also check whether the next digit is random. This led to the introduction of 2 more columns: “t+1_random” and “t+1_not_random” which also kept track of the results so far with integer values. By checking 2 digits simultaneously, the algorithm could also continuously check whether or not 2 digits in a row were considered to be random noise. If this is indeed the case, then all digits after these 2 random digits could be considered as insignificant. If the final digit of the float is reached without finding 2 random digits in a row, then one would only need to check whether the last digit is random and remove it if so. Additionally, although for this project the algorithm would only check whether 2 digits in a row are random, it would be relatively straightforward to turn the amount of checked digits into a parameter to check whether X digits in a row are random instead.

As stated before, for every interval, the algorithm would check whether a digit could be considered as random noise in that specific interval. In order to check whether this is or is not the case, it is important to keep in mind that every number that is possible for a single digit (0 to 9) would occur roughly 10% of the time if the digit is truly random noise. By calculating the true occurrence ratios for every possible number, the mean absolute error was then calculated. An “allowed mean absolute error” parameter was then added to the algorithm so that it could check whether the obtained mean absolute error is smaller than or equal to the allowed mean absolute error. If this is the case, then the digit is random. If the obtained error is greater than the allowed error, then the digit is not random. After these values have been calculated and compared for a specific interval, the columns that keep track of the results so far are updated and the interval moves forward.

4f. Implementing Bit Reduction

Once the amount of used digits had been reduced for many of the stored float values, the amount of bits used to store these values also needed to be reduced. After all, simply removing digits from floating point values does not reduce the amount of required storage (as explained in section 4b). For the bit reduction, it was crucial that the value of the number would not change. In order to achieve this goal, a function was created to would check whether or not the value of the float would remain constant if the number of bits were to be lowered.

Firstly, 3 empty lists and 1 dictionary would be created, one list for each possible amount of bits (16, 32 and 64). The lists would be filled by the row numbers of the values that would be

changed to use the amount of bits linked to their respective list. The dictionary on the other hand would be created as follows: {“float16”: 0, “float32”: 0, “float64”: 0}. Every time a row number is added to one of the lists, the dictionary’s values would change accordingly to keep track of how many values will use each possible specific amount of bits.

For every value in the dataset, the function checks whether the value remains constant if 16 bits were to be used rather than 64, since using 16 would take up the least amount of memory and would thus be the ideal situation when it comes to memory usage. If this is the case, then the created float16 list and the dictionary will be updated. If this is not the case, then the function will check whether the value can be stored with 32 bits without changing it. If this is also not the case, then 64 bits must be used for the value and the amount of bits can therefore not be reduced for that specific row.

The initial proposed methodology stated that the dataset would then be changed so that every row in the relevant column used the suitable amount of bits. However, this presented a problem since the Pandas library does not allow columns to contain multiple data type objects. This means that if all values are saved into the dataframe with the chosen amount of bits for every rows respectively, then the Pandas library would automatically use 64 bits for all rows which is undesirable. To solve this issue, 3 separate dataframes were created, one for each possible amount of bits. This way, the relevant column for each one of these dataframes only has a single data type object. Furthermore, both datasets that are used have an easy way to determine the chronological order (the OWEZ data uses multiple columns and the weather dataset uses the index). Therefore, it is not difficult to recombine these dataframes in the correct order when the dataset no longer needs to be compressed.

4g. Runtime

In order for the designed algorithm to be useful in a more practical setting, it is of critical importance that the runtime of said algorithm be acceptable. The runtime of the algorithm developed for this project has gone through many changes and the total time required to run the algorithm has been reduced drastically.

Once the first version of the algorithm had been finished, it immediately became clear that the runtime would be an issue. The code was allowed to run for one hour and based on the progress it had made in that time, a prediction was made as to how long it would take to run the entirety of the algorithm. This resulted in the prediction that the algorithm would take roughly 140 hours to finish running over the OWEZ dataset (which contains 252,632 rows after pre-processing). Clearly, that would not be an acceptable runtime. Therefore, some changes were made to the code concerning how the interval would progress through the dataset. In the first version of the algorithm, an interval was constructed (the length of which is determined by the interval parameter) that checks whether a certain digit in said interval can be considered as random-noise. This interval would move down by a single row after it had finished categorizing digits as random or non-random. Since the interval was quite large (for the OWEZ dataset it was usually equal to 10,000), it was unnecessary to move only one single row every time the interval moved. Therefore, it was changed so that the interval would move 100 rows after every iteration rather than 1. This could make the

analysis slightly less accurate, but it made the code run drastically faster. Only if the size of the interval were to be greatly reduced would this change have the potential to cause issues. However, since (within the scope of this project) there seems to be no reason to decrease the interval size by such a degree, this change did not present any issues. After this change, the runtime of the algorithm on the OWEZ dataset was roughly 1 hour and 20 minutes.

Though 1 hour and 20 minutes would be reasonably acceptable as a runtime, it is certainly not ideal. Luckily, there were still quite a lot of small changes that could be made to the algorithm to further improve the runtime. The most common way that runtime was further reduced at this point was by implementing built-in functions from certain libraries (most often the Pandas library). Although these functions served the same purpose and yielded the same output as my own code, the runtime was noticeably shorter for the built-in functions. Additionally, there were some situations where the algorithm would repeat a piece of code to calculate a value that had previously already been calculated and could be reused. By also ensuring that there would be as little repetition as possible, the runtime further decreased. After these steps, the runtime for the OWEZ dataset had been decreased to roughly 37 minutes. For the weather dataset on the other hand, the final runtime was roughly 2 minutes. This is because the weather dataset contains only 44,744 rows.

5. Results

The results of the performed analyses will be displayed and explained in this section.

5a. Random Noise Detection Results

As explained in section 4e, there are multiple parameters that can be altered in the Random Noise Detection algorithm. Firstly, there is the 'interval' parameter, which determines the size of the range of numbers that are considered when judging whether digits are random. Secondly, there is the 'allowed mean absolute error' parameter (or 'allowed MAE' for short). This parameter determines the maximum mean absolute error value for digits to be considered non-random. Lastly, there is the parameter 'Q', which will remain constant at a value of 1/3 as explained in section 4e.

The inclusion of these parameters imply that the algorithm could yield different result based on different values of these variables. Before the effects of these parameters will be explored and explained, the results of the algorithm will be shown for the parameter values that were used during the creation of the algorithm. These values were judged to be the best fit for the datasets that were being reduced:

Allowed MAE = 0.02
Interval = 5,000 for the Weather dataset
Interval = 10,000 for the OWEZ dataset

For the weather dataset the following distribution of float types resulted from the algorithm:

Float16: 1208 (2.70%)
Float32: 39050 (87.27%)
Float64: 4486 (10.03%)

These results show that the vast majority of floating point values are stored using 32 bits once the algorithm has finished. These changes have caused a clear reduction in the total amount of memory required for the columns. Whereas the relevant columns in the original weather dataset (after pre-processing) required 357,952 bytes to be stored. The columns only required a total of 194,504 bytes. This means that the required amount of memory has been decreased by 45.66%.

The distribution of float types for the OWEZ dataset on the other hand looked as follows:

Float16: 227,476 (90,04%)
Float32: 3 (0.001%)
Float64: 25,153 (9.96%)

These results would suggest that compared to the weather dataset, a far larger portion of the OWEZ dataset has been changed so that only 16 bits are used. The fact that only 3 rows make use of 32 bits is quite unexpected though. The number of rows that would make use of 32 bits was expected to be far larger, like it was the case in the weather dataset. The total amount of required bytes changed for the relevant column changed from a total of 2,021,056 bytes to only 656,188 bytes. This is a reduction of 67.53%.

5b. Effects of Parameters

In order to show more results yielded by the Random Noise Detection algorithm, the results of multiple combinations of parameters will be shown. The tables below display the effects of changing these parameters for both used datasets. The numbers in the tables represent the total number of bytes required for the column for which noise was detected and removed and the number of bits that have been reduced.

Weather Dataset:

	Interval = 2,500	**Interval = 5,000**	**Interval = 10,000**
Allowed MAE = 0.01	193,676	197,176	**202,864**
Allowed MAE = 0.02	192,544	194,504	202,864
Allowed MAE = 0.04	**190,144**	190,604	190,432

OWEZ Dataset:

	Interval = 5,000	**Interval = 10,00**	**Interval = 20,00**
Allowed MAE = 0.01	**656,366**	656,296	656,188
Allowed MAE = 0.02	656,074	656,188	656,164
Allowed MAE = 0.04	**655,166**	655,826	656,040

Within these tables, the cell with the lowest number of required bytes has been given a green colour. The cell with the largest number of required bytes on the other hand has been given a red colour. However, it should be noted that a lower amount of bytes is not always desirable as will be explored in the discussion section of this paper. The tables above show that an increase in allowed MAE causes the total required amount of memory to decrease. This makes sense since an increase in allowed MAE causes more digits to be categorized as random noise. This in turn causes the removal of more digits which may allow the use of fewer bits to store the floating point value. Therefore, an increase in allowed MAE could not possibly cause an increase in the memory usage. Changes in the interval parameter on the other hand are less straightforward to interpret. The results on the weather dataset seem to suggest that an increase in the interval parameter generally causes an increase in the required memory. However, the results of the OWEZ dataset seem to suggest that an increase in the interval parameter certainly do not always cause the required memory to increase.

5c. Evaluation of Hypotheses

Now that the results of the project have been presented, the hypotheses from section 4a will be evaluated. The hypotheses of were formulated as follows:

1. If the amount of **digits** used to save a floating point value decreases, the runtimes of algorithms which make use of said floating point value and the amount of memory usage required for said floating point value also decrease
2. If the amount of **bits** used to save a floating point value decreases, the runtimes of algorithms which make use of said floating point value and the amount of memory usage required for said floating point value also decrease
3. Performing bit reduction after random noise detection on the used datasets will reduce the memory usage of the datasets by at least 50%

Hypothesis 1 has been shown to be false. Decreasing the amount of digits does not have a positive effect when it comes to reducing memory usage. This is because the memory usage of every value in a dataset depends on the amount of bits used for said value. Reducing the amount of digits does not change the amount of used bits and therefore the total memory required to store a floating point value does not decrease if the amount of digits is decreased.

Hypothesis 2 on the other hand has been shown to be true. The storage and runtime analyses presented in section 4c clearly showed that saving floating point values with fewer bits does in fact lead to a decrease in memory usage.

Hypothesis 3 would strictly speaking be false. This is because the hypothesis states that the memory usage for both datasets would need to decrease by 50% or more. However, it could be argued that the hypothesis does hold true for one of the datasets. The results have shown that the size of the OWEZ dataset has decreased by 67.53%. However, they have also shown that the required memory for the weather dataset has decreased by 45.66%. This means that it depends on the specific dataset (and the chosen values for the parameters) whether or not the size of the dataset can be reduced by 50% or more.

Even though hypotheses 1 and (arguably) 3 are incorrect, finding out that they are incorrect could not only lead to a better understanding of the topic, but it could also lead to the formulation of relevant and valuable follow-up research questions and hypotheses.

6. Discussion

Although the results have been shown and explained, they have not yet been interpreted. In order for research to be truly useful, the implications of the results have to be discussion in detail so that they can be put into perspective. This may lead to better understanding what caused the results to be a certain way.

6a. Interpretation of Results

When looking at the results in the previous section, one may ask why the combination of parameters that leads to the lowest amount of required memory is not the best combination. After all, the purpose of this research is to minimize the amount of memory required to store datasets. Usually it would be good to reduce the size of a datasets even further. Within the context of this specific technique though, it could lead to non-noise data being categorized as random noise. This could well lead to relevant data being removed entirely from the dataset which could lead to massive problems. For example, one could imagine a dataset used in an architectural setting for the construction of a building. If relevant non-noise data has accidentally been removed from said dataset, this could lead to miscalculations that could well cause stability and safety concerns for the entire building. Should a building collapse due to relevant data being removed, many people could be endangered. There are many similar examples of dangers that would be the (in)direct effect of the unintended removal of non-noise data. Therefore, one should not consider the combination of parameters that lead to the largest reduction in data size to be the most suitable combination.

The resulting distribution of float types for the OWEZ dataset was unexpected to say the least. The extremely low ratio of values that were converted to the type float32 combined with the ~9% of values that still require 64 bits seems out of place. What makes this even more unexpected is that for the weather dataset, float32 was by far the most common type once the algorithm had finished. The low number of values stored with 32 bits could be caused by the value for the allowed MAE parameter being too large. This could lead to nearly all values that should be stored using 32 bits, being stored with 16 bits instead. However, this explanation would only make sense if the values that remained in the float64 category have certain digits that simply cannot be considered to be noise even if the value for allowed MAE is too large. After all, if that would not be the case, then the float32 category would still be filled by the values that change from using 64 bits to using 32 bits. However, additional experiments would need to be performed to find out with certainty whether or not this is indeed the reason. An alternative hypothesis for this phenomenon would be that all values in the OWEZ dataset can be stored using 16 bits. This would mean that certain digits are considered to be non-noise while they actually are noise.

Another oddity within the obtained results is that the required memory for the OWEZ dataset does not change much when the values of the parameters are altered. This is especially true when comparing the results for the OWEZ dataset with the results for the weather dataset. The results for the weather dataset seem to change far more when parameter values change. The most likely explanation would once again be that the value for the allowed MAE is too large. If the value of allowed MAE is too large, then very few values will be changed to use fewer bits as the allowed MAE increases further. This is because there would already be too many digits that are considered to be noise even though they may not be noise at all. This would cause changes in required memory to be quite small when parameters change in value. This possible explanation is further supported by the fact that the OWEZ dataset's size is reduced far more than the weather dataset. Whereas the OWEZ dataset's size decreases by 67.53%, the weather dataset's size decreased by only 45.66%. This could be a sign that too many digits in the OWEZ dataset are being categorized as noise.

6b. Ethical & Legal Considerations

When research leads to the development of new technology, ethical and legal aspects need to be considered. The reason for this being that new technologies can often cause unforeseen issues for users or society as a whole. Considering aspects of new technology that could lead to these kind of issues before said technology is used in a practical setting can avoid a lot of (either financial or physical) harm. When it comes to the algorithm developed in this project, two aspects have been identified that could potentially lead to issues.

The first of these aspects is the possible outcomes if the algorithm makes faulty classifications. Digits can sometimes be classified as random noise when they are in fact not the results of random noise. This could lead to significant data being removed from the dataset. One could imagine an engineer using the developed algorithm to compress a dataset. If this were to lead to significant information being lost, calculations performed by the engineer could end up being faulty as well. As a result, a product that was developed with the use of the faulty calculations could very well malfunction which would lead to financial and possibly physical harm. To avoid this, it is important to emphasise that the developed algorithm is a lossy compression algorithm. Even though the algorithm is centred around the removal of random-noise, the classification will not be correct 100% of the time. This means that small portions of significant data may unintentionally be removed occasionally.

The second aspect of the algorithm that could lead to issues is the question of who would be considered to be responsible for harm that results from misclassifications by the algorithm. This mishandled, this aspect could lead to serious ethical and legal issues. If one considers the example provided in the previous paragraph, one could ask themselves whether the harm was caused by the engineer or by the algorithm. It could be said that the answer depends on the level of communication by the algorithm developer. If the developer does not communicate the possible disadvantages and what the algorithm can and cannot be used for, then one could argue that the developer would be responsible for harm caused by their algorithm. However, what if the developer is as transparent and open about the

algorithm as possible and communicates the purpose and possible disadvantages for the algorithm well? One could argue that this would be an entirely different situation in which the user of the algorithm could be considered to be responsible for misusing the algorithm. That is why proper communication can be considered to be the key to these ethical and legal considerations.

7. Conclusion

Now that the results of this project have not only been shown and explained, but also discussed, the final conclusions will be presented. As shown in the results section of this paper, the data reduction algorithm has successfully reduced the size of two datasets by a reasonable amount. Whereas the weather dataset decreased in memory usage by 45.66%, the memory usage of the OWEZ dataset decreased by 67.53%. This decrease in memory usage was achieved by only removing random-noise digits and by minimizing the amount of bits used for every entry in the used datasets. There are lossy compression methods that achieve larger decreases in memory usage than this methods. However, these lossy compression methods are more likely to remove relevant data since the algorithm developed in this project was created to only remove data that does not add any additional useful information. For the random noise detection and bit reduction algorithms developed for this project to truly be useful in a practical setting however, they could be combined with existing compression techniques to reduce the size of datasets even further.

7a. Follow-up Research

Though this project has yielded many valuable insights, there are still some questions that have not (fully) been answered. Unfortunately, it is impossible for one paper to answer all questions about a certain topic. This is where follow-up research can be of help. In the future, other members of the scientific community may be able to find answers for the questions that have been left unanswered in this paper. There are 5 main topics relating to this project that could be explored in follow-up research.

Firstly, there is the question of what the optimal combination of parameters would be. In this project, the algorithm takes 2 parameter values and returns the required memory to store the dataset afterwards. A problem with this approach is that the output does not clearly show whether the effect of changes in parameters is positive or negative. This is because less required memory is not necessarily a good thing if it leads to non-noise data being removed from the dataset. It could be interesting to see research catered around finding the optimal combination of parameters. It could be possible for that an algorithm could be developed by researchers with the appropriate domain knowledge to (perhaps systematically) find the optimal combination of parameters for different datasets.

Secondly, follow-up research could be conducted concerning the usage of F tests (or other hypothesis tests) for data size reduction. Although this project did not manage to reduce the

sizes of the used datasets with the use of F tests, it did yield some insights that could be useful for those with the statistical expertise to continue where this project left off.

Thirdly, it could certainly be possible for machine learning algorithms to be developed for the detection of random-noise digits. However, to train a prediction model for this task, one would need to have a dataset in which every digit is marked as random or non-random. This would be necessary so that the prediction model can be trained on a training dataset. If this could lead to the development of an algorithm that more accurately predict whether digits are random noise, it could make the techniques described in this paper a lot more powerful and useful.

Fourthly, follow-up research could be conducted regarding the binary representation of floating point values. While this project has focussed on the decimal representation of floating point values, these values are more commonly represented in a binary way. Altering the created algorithm in such a way that it would make use of the binary representation could yield valuable insights that may improve the effectiveness of the created algorithm.

Lastly, there is one more topic of this project that could lead to follow-up research. In this project, if for an arbitrary value of X, the distribution of the X_{th} digit shows that all numbers from 0 to 9 are equally common, then the X_{th} digit is considered to be noise. However, there are situations where this may not be true. For example, if the first 10% of rows has the number 1 at the X_{th} digit, but after that, no rows have the number 1 at the X_{th} digit, that is very unlikely to be a coincidence. The X_{th} digit is therefore extremely unlikely to be random noise. However, the algorithm used for this project would not be able to detect these oddities and could very well categorize the X_{th} digit as random noise. Follow-up research into ways to detect these oddities could therefore be of great value.

7b. Answer to Research Question

As described in section 2d, the research question for this project was: “How can lossy compression methods be used to reduce the memory usage of a dataset without significant information being lost?”. To answer this question, one should first consider the two sub questions that were created to provide the required insights to answer it.

The first of these sub questions asked how random noise detection can be used to remove insignificant digits from quantitative data in order to reduce the required memory usage for datasets. In this paper, the answer to this question has been explained in detail. However, to sum the answer up in one sentence: By checking whether the mean absolute error of a digit’s distribution is smaller than a pre-determined allowed mean absolute error, random noise digits can be detected and removed so that the amount of bits used to store the values can be minimized, thus reducing the memory usage of the dataset.

The second sub question asked how F tests be used to remove insignificant digits from quantitative data in order to reduce the required memory usage for datasets. Unfortunately, this project did not yield any concrete methods to reduce data sizes with the use of F tests. However, the fact that this project did not find any viable data size reduction methods using

F tests does not mean these methods may not be developed in the future. The insights gained in this project concerning F tests can serve as the bedrock for follow-up research.

With the gained insights from these sub questions, the main research question of this project can now be answered. Noise reduction followed by bit reduction (as described in this paper) can be used to reduce the sizes of datasets while keeping the loss of non-noise data to a minimum. F tests on the other hand have not yielded any usable compression techniques within the scope of this project. It should be noted however that there are many possible compression techniques and the ones researched in this project only show a small part of the bigger picture.

In conclusion, random noise detection can be used to reduce the memory usage of datasets. This is done by removing digits with a random distribution and then minimizing the amount of bits used for values in the dataset. In combination with existing compression techniques, this method can be useful for the reduction of quantitative datasets containing floating point data.

References

Quaeghebeur, E. & Zaaijer, M.B. (2020) - How to improve the state of the art in metocean measurement datasets. *European Academy of Wind Energy, 5*(1), 285-308. https://doi.org/10.5194/wes-5-285-2020

Ur Rehman, M.H., Liew, C.S., Abbas, A. et al., (2016) - Big Data Reduction Methods: A Survey. *Data Sci. Eng., 1*(1), 265–284 (2016). https://doi.org/10.1007/s41019-016-0022-0

Mairs, S. et al., (2015) - The JCMT Gould Belt Survey: a quantitative comparison between SCUBA-2 data reduction methods. *Monthly Notices of the Royal Astronomical Society, 454*(3), 2557-2579. https://doi.org/10.1093/mnras/stv2192

Cao, H., Wolfson, O., Trajcevski, G. (2006) - Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal, 15*(1), 211-228. https://doi.org/10.1007/s00778-005-0163-7

Goldschneider, J.R. (1997) - *Lossy compression of scientific data via wavelets and vector quantization*. University of Washington. https://www.proquest.com/openview/574796c3ea765da0275abde74c8d0734/1?pq-origsite=gscholar&cbl=18750&diss=y

Kosakovsky Pond, S.J., Frost, S.D.W., Muse, S.V. (2004) - HyPhy: hypothesis testing using phylogenies. *Bioinformatics, 21*(5), 676-679. https://doi.org/10.1093/bioinformatics/bti079

Nickerson, J.A., Sloan, T.W. (1999) - Data reduction techniques and hypothesis testing for analysis of benchmarking data. *International Journal of Production Research, 37*(8), 1717-1741. https://doi.org/10.1080/002075499190978

Namey, E., Guest, G., Thairu, L., Johnson, L. (2008) - Data Reduction Techniques for Large Qualitative Data Sets. *Handbook for team-based qualitative research*, *2*(1), 137-161. http://qualquant.org/wp-content/uploads/cda/Namey%20and%20Guest%20Data%20Reduction.pdf

Cappello, F., Di, S., Li, S., Liang, X., Gok, A.M., Tao, D., Yoon, C.H., Wu, X.C., Alexeev, Y., & Chong, F.T., (2019) - Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications, 33*(6), 1201-1220. http://doi.org/10.1177/1094342019853336

Marzen S.E, De Deo S. (2017) - The evolution of lossy compression. *Journal of The Royal Society Interface, 14*(130), 130-166. https://doi.org/10.1098/rsif.2017.0166

Allisy-Roberts, P., Williams, J. (2008) - *Farr's Physics for Medical Imaging (Second Edition).* Saunders Ltd. https://doi.org/10.1016/C2009-0-34335-4

Xiong, Z., Wu, X., Cheng, S., Hua, J. (2003) - Lossy-to-Lossless Compression of Medical Volumetric Data Using Three-Dimensional Integer Wavelet Transforms. *IEEE Transactions on Medical Imaging, 22*(3), 459-470. https://doi.org/10.1109/TMI.2003.809585

Said. A., Pearlman, W. (1996) - An Image Multiresolution Representation for Lossless and Lossy Compression. *IEEE Transactions on Image Processing, 5*(9), 1303-1310. https://doi.org/10.1109/83.535842

Mentzer, F., Van Gool, L., Tschannen, M. (2020) - Learning Better Lossless Compression Using Lossy Compression. *Proceedings of The IEEE/CVF Conference on Computer Vision and Pattern Recognition,*

2020(1), 6638-*6647.*
https://openaccess.thecvf.com/content_CVPR_2020/papers/Mentzer_Learning_Better_Lossless_Compression_Using_Lossy_Compression_CVPR_2020_paper.pdf

Goyal, V., Fletcher, A.K., Rangan, S. (2008) - Compressive Sampling and Lossy Compression. *IEEE Signal Processing Magazine, 25*(2), 48-56. https://doi.org/10.1109/MSP.2007.915001

Underwood, R., Di, S., Calhoun, J., Cappello, F. (2020) - FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data. *IEEE International Parallel and Distributed Processing Symposium, 2020*(1), 567-577.
https://doi.org/10.1109/IPDPS47924.2020.00065

Chen, Y. et al., (2014) - DaDianNao: A Machine-Learning Supercomputer. *47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014*(1), 609-622.
https://doi.org/10.1109/MICRO.2014.58

Bell, G. (2014) - *Supercomputers: The amazing race (a history of supercomputing, 1960-2020)*. TechReport MSR-TR-2015-2. Ver. 1. 555 California, 94104 San Francisco, CA: Microsoft Corporation.
http://gordonbell.azurewebsites.net/msr-tr-2015-2_supercomputers-the_amazing_race_bell.pdf

McCalpin, J. (2016) - Memory bandwidth and system balance in HPC systems. *UT Faculty/Researcher Works.*

Vajda, A. (2011) - Multi-core and Many-core Processor Architectures. *Programming Many-Core Chips. Springer, Boston, MA*, 2011(1), 9-43. https://doi.org/10.1007/978-1-4419-9739-5_2

Bautista-Gomez, L., Zyulkyarov, F., Unsal, O., McIntosh-Smith, S. (2016) - Unprotected Computing : A Large-Scale Study of DRAM Raw Error Rate on a Supercomputer. *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016*(1), 645-655. https://doi.org/10.1109/SC.2016.54

Sridharan, V., et al., (2013) - Feng Shui of Supercomputer Memory Positional Effects in DRAM and SRAM Faults. *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013*(1), 1-11.
https://ieeexplore.ieee.org/abstract/document/6877455

Jung, L. (2007) - Writing SMART Objectives and Strategies That Fit the Routine. *TEACHING Exceptional Children, 39*(4), 54-58. https://doi.org/10.1177/004005990703900406

Ogbeiwi, Osahon. (2017) - Why written objectives need to be really SMART. *British Journal of Healthcare Management, 23*(7), 324-336. https://doi.org/10.12968/bjhc.2017.23.7.324

Moreira, E.E., Mexia, J.T., Minder, C.E. (2013) - F tests with random samples size. Theory and applications. *Statistics & Probability Letters, 83*(6), 1520-1526.
https://doi.org/10.1016/j.spl.2013.02.020

Poisson, S.D. (1837) - *Recherches sur la probabilité des jugemens en matière criminelle at en matière civile, précédés des règles générales du calcul des probabilités*. Adament Media Corporation.

Seneta, E. (2013) - A Tricentenary history of the Law of Large Numbers. *Bernoulli, 19*(4), 1088-1121.
https://doi.org/10.3150/12-BEJSP12

Soliman, A. (2020) - Laws Of Large Numbers And Khinchin's Constant.
http://math.uchicago.edu/~may/REU2020/REUPapers/Soliman.pdf

Chen, Z., Wu, P., Li, B. (2013) - A strong law of large numbers for non-additive probabilities. *International Journal of Approximate Reasoning, 53*(3), 365-377. https://doi.org/10.1016/j.ijar.2012.06.002

Seo, S. (2006) - *A review and comparison of methods for detecting outliers in univariate data sets.* Doctoral dissertation, University of Pittsburgh. http://d-scholarship.pitt.edu/7948/

Higham, N., Pranesh, S. (2019) - Simulating low precision floating-point arithmetic. *SIAM Journal on Scientific Computing, 41*(5), 585-602. https://doi.org/10.1137/19M1251308

Haidar, A., Tomov, S., Dongarra, J., Higham, N. (2018) - Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers. *SC'18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018*(1), 603-613. https://doi.org/10.1109/SC.2018.00050

Werle Lee, K.P. (2010) - Planning for success: setting SMART goals for study. *British Journal of Midwifery, 18*(11), 744-746. https://doi.org/10.12968/bjom.2010.18.11.79568

Huber, M.F., Haneback, U.D., (2008) - Gaussian Filter based on Deterministic Sampling for High Quality Nonlinear Estimation. *IFAC Proceedings Volumes, 41*(2), 13527-13532. https://doi.org/10.3182/20080706-5-KR-1001.02291

Mathys, C.D., et al., (2014) - Uncertainty in perception and the Hierarchical Gaussian Filter. *Frontiers in Human Neuroscience, 8*(1), 825. https://doi.org/10.3389/fnhum.2014.00825

Pence, W.D., White, R.L., Seaman, R. (2010) - Optimal compression of floating-point astronomical images without significant loss of information. *Publications of the Astronomical Society of the Pacific*, *122*(895), 1065. https://doi.org/10.1086/656249

Ying, X. (2019) - An overview of overfitting and its solutions. *Journal of physics: Conference series, 1168*(2), 22022. http://doi.org/10.1088/1742-6596/1168/2/022022