Investigations on the Computation and Results of an Integral Distance on Hyperspaces

Author: Darren Schmidt

Advisor: Dr. Matt Insall

Department: Mathematics and Statistics

April 1, 2021

Abstract

In this paper, we will be investigating how to compute the integral distance defined in [1], and we will analyze the results from this computation. We develop a way to compute the integral distance by using Monte-Carlo Integration, and we analyze the time complexity and the error that results from this method of computation. We also investigate when this distance function is a metric, how this metric compares to some other common metrics, and what the efficacy of the integral distance is for computer vision problems.

1 Introduction

The integral distance in [1] defines a metric that computes distances between sets when $p \geq 1$. That is, if X is a set, then the metric computes a distance between two elements of 2^X. In this paper, we give detail about how we approximate this metric by using Monte-Carlo Integration.

In [1], the integral distance is defined on the compactum X, with $A, B \subset X$ being closed, and the function $d_A(x) = \inf\{d(x, y)|y \in A\}$. The paper then defines the integral metric as

$$D_p(A, B) = \left(\int_X |d_A(x) - d_B(x)|^p dx \right)^{\frac{1}{p}}.$$

Defined in [[2], p. 11], Monte-Carlo Integration is a numerical integration algorithm that relies on statistical sampling of values of the integrand to estimate the value of the integral. Due to Monte-Carlo Integration's statistical basis, the algorithm can be used to find a confidence interval for the integral for a given confidence level. The benefit of this numerical analysis technique is that the error is only dependent on the number of sample points, and does not change as the dimension changes, as some other integration algorithms do. Specifically, if we let s represent the number of sampling points of the integrand, then the error of the Monte-Carlo algorithm scales with $\frac{1}{\sqrt{s}}$ [[3], p. 12]. Since we are investigating the integral metric on sets in the power set of $\mathbb{R}^n$, this is useful, as the error of the integral approximation will not be dependent on n. The algorithm is listed in section 6.5

To analyze the Monte-Carlo algorithm, we created a program that can generate thousands of sets of real numbers, seen in section 6.5 to see what distance the integral distance gives when plugging in those sets. The data that we generate in this program comes from a pseudorandom number generator provided by Python, as a computer cannot generate truly random data. We will still say that this data is random, as is common in the applied literature. We can use this program to examine what values the integral distance gives for different types of sets. This can help us understand how the metric behaves, and whether it can be reliably used for other purposes. These other purposes include using it for image analysis, among other things. Some classification algorithms rely on a metric to classify data, and we want to see how this metric compares with those other classification algorithms.

We utilize the results from the numerical approximation to compare the integral distance to other metrics, including the Euclidean Metric and the Hausdorff metric. Since the integral distance is provably a metric on hyperspaces, to compare it to the Euclidean metric, we only look at the distance between singleton sets. The Hausdorff metric is also a metric on hyperspaces, so the metrics can be compared for all sets. We use the numerical approximation of the integral metric to determine how the integral metric relates to other metrics.

In [1], the integral distance is only proved to be a metric when $p \geq 1$. The authors prove that for all $p > 0$, the integral distance is a semimetric, but they only prove that it satisfies the triangle inequality when $p \geq 1$. In this paper, we investigate whether the distance function satisfies the triangle inequality when $0 < p < 1$. Specifically, we prove that the integral distance under the usual distance function in $\mathbb{R}$ does satisfy the triangle inequality when $0 < p < 1$, and is therefore a metric.

When the integral distance is a metric, it can be utilized in various computer vision algorithms that use distances to distinguish objects. We investigate its efficacy in performing this function by analyzing how well it can read the data in the MNIST data set. The MNIST data set contains 70,000 handwritten entries of the numbers zero through nine, each transformed into a twenty-eight by twenty-eight pixel image. The goal is to try to correctly classify as many numbers as possible. We attempt this goal by using the k-medoids PAM algorithm, as detailed in [4].

2 Time Complexity and Error Analysis

We created a program to generate a large amount of randomly generated data that could be substituted into the algorithm for approximating the integral metric. We used these values to analyze the time complexity of the program and how the error changes based on the changing the parameters of the program. Plots for these values are included in the Appendix in 6.1 and 6.2. Throughout this investigation, we found that the time complexity of the approximation algorithm increases linearly when the cardinalities of the sets A and B substituted into the integral metric increases, when the number of points that are sampled increases, and when the dimension increases. The edge length of the set X does not affect the time complexity, where the edge length of $X = [x_0, x_1]^n$ is $x_1 - x_0$. If n is the number of sampled points, then as n increases, the computation error decreases by $\frac{1}{\sqrt{n}}$, which is consistent with how the Monte-Carlo Integration algorithm should work. The error is constant when varying the cardinalities of the sets A and B. However, the error increases at a rate faster than linear when the length of the set X increases, and the error seems to increase at least exponentially as the dimension increases. These results were determined by varying only one parameter at a time.

In the introduction, we stated that the error for approximating integrals using Monte-Carlo Integration is dependent only on the number of points being sampled for the integrand. The above result states that the error also increases when dimension increases and when the length of the set X increases. In the case of dimension increasing, this happens due to the integrand itself, not due to the integration method. The integrand is dependent on the Euclidean metric, which will have another term to compute whenever the dimension increases by one. This change in the function when the dimension changes affects the error when sampling points of the function to approximate the integral. The error is not the result of the Monte-Carlo algorithm. With an increase in the length of the set X, there are additional values that can be substituted into the integrand, changing the codomain of the integral. This means that there are more points that can be sampled to approximate this integral. Since the number of sample points was kept constant, this means that increasing the edge length of X adds more values that were not being considered in the approximation of the integral, increasing the error.

3 Numerical Approximation Results

Generating large amounts of data for the Monte-Carlo method, we analyze thousands of computations or more within a few seconds to measure how the integral metric performs with various types of data. For instance, we can fix the integral to calculate 1,000 different values of the integrand, and then see how the results change if we change the set X to be different intervals of $\mathbb{R}$. We thereby test the correctness of the algorithm, and we analyze the time complexity and error depending the type of data entered into the program.

Utilizing this numerical approximation algorithm for the integral metric, we can use this program to test how the integral metric performs in various data analysis algorithms for established data sets. The K-Nearest neighbors algorithm, a supervised learning algorithm, and the k-medoids algorithm, an unsupervised learning algorithm, both use a metric for clustering and classifying data. Using the integral distance as the metric in these algorithms, we extract useful information from data sets, such as the MNIST handwritten digits data set. We plan on using it to analyze other image classification problems. Since our algorithm approximates values of the integral distance, we can use the algorithm to calculate values of the metric needed for the data analysis algorithms.

4 Comparison with Other Metrics

4.1 Metric Comparisons

We use the Monte-Carlo algorithm to compare the integral metric with the Euclidean metric and the Hausdorff metric. We study the question of whether the metrics are strongly equivalent. Two metrics d_1 and d_2 defined on a non-empty set X are said to be strongly equivalent metrics, as stated in [[3], p. 86], if $\exists c_1, c_2 > 0$ such that $c_1 d_1(x,y) \leq d_2(x,y) \leq c_2 d_1(x,y), \forall x, y \in X$. The program cannot prove the existence of such bounds, but through computations we can estimate what these bounds may be.

The Euclidean metric is a metric on points, not on sets like the integral metric. Therefore, we reduce the domain of the integral metric to singleton sets, and compare integral metric results for the distance between the singleton sets to Euclidean metrics results for the distance between the points in those sets. Computation results in 6.4 suggest an upper bound, dependent on the space $X = [x_0, x_1]^n \subseteq \mathbb{R}^n$, of $(x_1 - x_0)^{\frac{n}{p}}$. This is proven in section 4.2 for $\mathbb{R}^1$. A lower bound was not found using this program, we prove this for $\mathbb{R}^1$ in section 4.2. We have been exploring the results from this program to try to find lower bounds for $\mathbb{R}^2$ and higher dimensional spaces.

We also use the Monte-Carlo algorithm to compare the integral metric to the Hausdorff metric, using a predefined function for the Hausdorff metric in the scipy python package. This follows the definition of the Hausdorff metric defined in [[5], p. 53]. As described in [5], the Hausdorff metric is a metric on hyperspaces, just like the integral metric, so it returns a distance between sets. We are continuing to investigate whether the algorithm leads to possible bounds to prove this is strongly equivalent to the integral metric, or whether the algorithm provides a counterexample that the bounds do not exist and the two metrics are not strongly equivalent.

4.2 Results

The following theorem is a consequence of corollary 3.1 in [[1], p. 148] when looking at singleton sets.

Theorem 1. *Let $A = \{a\}$, $B = \{b\}$, such that $a, b \in \mathbb{R}$. Let $X = [c,d] \subseteq \mathbb{R}$ such that $c \leq a \leq b \leq d$. Also, let $d(x,y)$ be the Euclidean metric, and let $D_p(A,B)$ be the p-th order integral metric. Then, $D_p(A,B) = d(a,b)(d - c - \frac{p|a-b|}{p+1})^{\frac{1}{p}}$ when $p > 0$.*

Proof. In this case of $\mathbb{R}^1$, where $A = \{a\}$ and $B = \{b\}$ are singleton sets,

$$D_p(A,B) = \left(\int_c^d ||x-a| - |x-b||^p dx \right)^{\frac{1}{p}}.$$

We will now examine the integrand, which we will denote as $f(x) = ||x-a|-|x-b||^p$. The derivative f is given by is $f'(x) = p||x-a|-|x-b||^{p-1}(\frac{|x-a|-|x-b|}{||x-a|-|x-b||})(\frac{x-a}{|x-a|} - \frac{x-b}{|x-b|})$. By setting this equal to zero, we can find the critical points, and the trivial critical points are at $x = a$ or $x = b$. There is also another critical point at $x = \frac{a+b}{2}$. Now, since $||x-a|-|x-b||^p = ||x-b|-|x-a||^p$, we can assume without loss of generality that $a \leq b$. Let $\epsilon > 0$, and let $x = a - \epsilon$. We then see that $f(x) = f(a-\epsilon) = ||a-\epsilon-a|-|a-\epsilon-b||^p = |\epsilon+a-\epsilon-b|^p = |a-b|^p$. Now, let $x = b + \epsilon$, then $f(b+\epsilon) = ||b+\epsilon-a| - |b+\epsilon-b||^p = |b-a+\epsilon-\epsilon|^p = |a-b|^p$. By letting $x = a$ or $x = b$, we see that $f(a) = ||a-a| - |a-b||^p = |a-b|^p$, and $f(b) = ||b-a| - |b-b||^p = |a-b|^p$, so $f(a) = f(b)$ is either a global minimum or global maximum of this function. If we let $0 < \epsilon < b - a$, and we let $x = a + \epsilon$, then $f(x) = f(a+\epsilon) = ||a+\epsilon-a|-|a+\epsilon-b||^p = |\epsilon-|a-b+\epsilon||^p = |\epsilon+a-b+\epsilon|^p = |a-b+2\epsilon|^p = |2x-a-b|^p$. For $a < x < b$, this is less than $|a-b|$, so $f(a) = f(b)$ is a global maximum for this function. Now, to analyze f, we first see when it equals 0, so this is the global minimum. This happens when $x = \frac{a+b}{2}$. Now, we claim that the function f is symmetric about this point; that is, $f(y + \frac{a+b}{2}) = f(\frac{a+b}{2} - y)\ \forall y \in X$. We start with the left hand side, where $f(x + \frac{a+b}{2}) = ||x + \frac{a+b}{2} - a| - |x + \frac{a+b}{2} - b||^p = |2x|^p$. The right hand side is $f(\frac{a+b}{2} - x) = ||\frac{a+b}{2} - x - a| - |\frac{a+b}{2} - x - b||^p = |2x|^p$. The claim follows. We also note that $|2x - a - b|^p > 0$

when $x > \frac{a+b}{2}$. Now, to evaluate the integral distance in this case:

$$D_p(A,B) = \left(\int_c^d ||x-a| - |x-b||^p dx\right)^{\frac{1}{p}} \tag{1}$$

$$= \left(2\int_{\frac{a+b}{2}}^b (2x-a-b)^p dx + (a-c)|a-b| + (d-b)|a-b|\right)^{\frac{1}{p}} \tag{2}$$

$$= \left(\frac{|a-b|^{p+1}}{p+1} + (a-c)|a-b|^p + (d-b)|a-b|^p\right)^{\frac{1}{p}} \tag{3}$$

$$= |a-b|\left(a-b+d-c+\frac{|a-b|}{p+1}\right)^{\frac{1}{p}} \tag{4}$$

$$= d(a,b)\left(d-c-\frac{p|a-b|}{p+1}\right)^{\frac{1}{p}}, \tag{5}$$

when $p > 0$.

□

Theorem 2. *When $p \geq 1$, the integral metric is strongly equivalent to the Euclidean metric on $X = [x_0, x_1] \subseteq \mathbb{R}$.*

Proof. Let $X = [x_0, x_1] \subseteq \mathbb{R}$, and let $A = \{a\}$ and $B = \{b\}$ such that $x_0 \leq a \leq b \leq x_1$. Then, Theorem 1 tells us that $D_p(A,B) = d(a,b)\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}}$ when $p > 0$. We need to find positive constants α and β such that $\alpha * d(a,b) \leq D_p(a,b) \leq \beta * d(a,b)\ \forall a,b \in \mathbb{R}$. This is equivalent to $\alpha \leq \left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}} \leq \beta$. The middle term has a maximum when $|a-b| = 0$, which implies that $\beta = (x_1 - x_0)^{\frac{1}{p}}$. The middle term has a minimum when $|a-b|$ is at its maximum. In this space, the maximum value is $|a-b| = x_1 - x_0$, so the minimum value is $\left(\frac{x_1-x_0}{p+1}\right)^{\frac{1}{p}}$. This means that $\alpha = \left(\frac{x_1-x_0}{p+1}\right)^{\frac{1}{p}}$. □

The following lemmas are used to prove Theorem 3, which states that when $0 < p < 1$, the integral distance with the standard distance in $\mathbb{R}$ satisfies the triangle inequality. Since [1] shows that the integral distance is always a semimetric, this means that the integral distance is a metric in this case.

Lemma 1. *Let $a,b,c \in \mathbb{R}$. If $b < c < a$ or $a < c < b$, then $|a-c| < |a-b|$. If $b < a < c$ or $c < a < b$, then $|a-c| < |b-c|$. If $a < b < c$ or $c < b < a$, then $|a-c| = |a-b| + |b-c|$.*

Proof. If $b < c < a$, then $b-a < c-a$, which means that $|a-c| < |a-b|$. If $a < c < b$, then $c-a < b-a$, which implies that $|a-c| < |a-b|$. If $b < a < c$. then $b-c < a-c$, which means that $|a-c| < |b-c|$. If $c < a < b$, then $b-c > a-c$, which implies that $|a-c| < |b-c|$. If $a < b < c$, then $|a-b|+|b-c| = b-a+c-b = |a-c|$. If $c < b < a$, then $|a-b| + |b-c| = a-b+b-c = |a-c|$. □

Lemma 2. *Let $a,b,c,x_0,x_1,p \in \mathbb{R}$. Let $x_0 < x_1$, $0 < p < 1$, $|a-c| < |a-b|$, and $|a-c|$,$|a-b| < x_1 - x_0$. Also, let $f(p) = \left(\frac{x_1-x_0-\frac{p|a-c|}{p+1}}{x_1-x_0-\frac{p|a-b|}{p+1}}\right)^{\frac{1}{p}}$. Then, f is monotone decreasing on $(0,1)$.*

Proof. We note that $\log(f(p)) = \frac{1}{p}\log\left(\frac{x_1-x_0-\frac{p|a-c|}{p+1}}{x_1-x_0-\frac{p|a-b|}{p+1}}\right)$, and that

$$\frac{1}{f(p)}f'(p) = \frac{p(x_1-x_0)(|a-b|-|a-c|) - (p+1)^2\left(x_1-x_0-\frac{p|a-c|}{p+1}\right)\left(x_1-x_0-\frac{p|a-b|}{p+1}\right)\log\left(\frac{x_1-x_0-\frac{p|a-c|}{p+1}}{x_1-x_0-\frac{p|a-b|}{p+1}}\right)}{p^2(p+1)^2\left(x_1-x_0-\frac{p|a-c|}{p+1}\right)\left(x_1-x_0-\frac{p|a-b|}{p+1}\right)}$$

We note that the denominator is positive, so the fraction has the same sign as the numerator. We also note that $f(p) > 0$, so $\frac{1}{f(p)} > 0$, and $f'(p)$ has the same sign as the numerator of this fraction. Now, denote the numerator as

$$g(p) = p(x_1-x_0)(|a-b|-|a-c|)-(p+1)^2\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)\log\left(\frac{x_1 - x_0 - \frac{p|a-c|}{p+1}}{x_1 - x_0 - \frac{p|a-b|}{p+1}}\right)$$

Now, let

$$\begin{aligned} h(p) &= \frac{g(p)}{(p+1)^2\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)} \\ &= \frac{p(x_1-x_0)(|a-b|-|a-c|)}{(p+1)^2\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)} - \log\left(\frac{x_1 - x_0 - \frac{p|a-c|}{p+1}}{x_1 - x_0 - \frac{p|a-b|}{p+1}}\right) \end{aligned}$$

We see that the derivative is given by

$$h'(p) = \frac{\alpha}{\beta}$$

Where

$$\begin{aligned} \alpha = (|a-c| - |a-b|)(x_1-x_0)\,p\,\big(p\,\big(2x_1^2 + x_1(-4x_0 - 2|a-c| - 2|a-b|) + 2x_0^2 + 2x_0(|a-c| + |a-b|) \\ + 2|a-c|\,|a-b|) + 2x_1^2 + x_1(-4x_0 - |a-c| - |a-b|) + 2x_0^2 + x_0(|a-c| + |a-b|)\big) \end{aligned}$$

$$\beta = ((x_1 - x_0 - |b-a|)\,p + x_1 - x_0)^2\,((x_1 - x_0 - |c-a|)\,p + x_1 - x_0)^2$$

We see that

$$(x_1-x_0)(|a-b| + |a-c|) < 2(x_1-x_0)^2,$$

so

$$2x_1^2 + x_1(-4x_0 - |a-c| - |a-b|) + 2x_0^2 + x_0(|a-c| + |a-b|) > 0.$$

We also see that

$$\begin{aligned} &2[(x_1-x_0)(|a-c| + |a-b|) - |a-c||a-b|] \\ &= 2[|a-b|(x_1 - x_0 - |a-b|) + (x_1-x_0)|a-b| \\ &< 2(x_1-x_0)(x_1-x_0) \\ &= 2(x_1-x_0)^2, \end{aligned}$$

which shows that

$$2x_1^2 + x_1(-4x_0 - 2|a-c| - 2|a-b|) + 2x_0^2 + 2x_0(|a-c| + |a-b|) + 2|a-c||a-b| > 0.$$

We see that $\beta > 0$, and since $(|a-c| - |a-b|) < 0$, $\alpha < 0$. That means that $h'(p) < 0$, and since $h(0) = 0$, $h(p) < 0$, then $g(p) < 0$ and $f'(p) < 0$. Therefore, for $0 < p < 1$, $f(p)$ is monotone decreasing.

□

Lemma 3. *Let $x, y, a \in \mathbb{R}$, where $0 < x,y \leq a$. Then, $\log(x) - \log(y) < \frac{1}{a}(x-y)$ implies that $x < y$.*

Proof. If $\log(x) - \log(y) < \frac{1}{a}(x-y)$, then $\frac{x}{a} - \log(x) > \frac{y}{a} - \log(y)$. Let $f(x) = \frac{x}{a} - \log(x)$. Then $f'(x) = \frac{1}{a} - \frac{1}{x}$. Consequently, f has no critical points on the interval $(0, a)$, and we see that it is decreasing on this interval. Since, $\frac{x}{a} - \log(x) > \frac{y}{a} - \log(y)$, it follows that $x < y$. □

Theorem 3. *Let $x_0, x_1, p \in \mathbb{R}$, $X = [x_0, x_1]$, and $0 < p < 1$. Let D_p be the integral distance restricted to the interval X with the Euclidean metric on $\mathbb{R}$. Then, D_p is a metric.*

Proof. Let $a, b, c \in X$. From [1], we know that D_p is a symmetric, so we only need to prove the triangle inequality. We know that $D_p(\{a\},\{b\}) = |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}}$. The triangle inequality $D_p(\{a\},\{c\}) \leq D_p(\{a\},\{b\}) + D_p(\{b\},\{c\})$ is therefore equivalent to stating that $|a-c|\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)^{\frac{1}{p}} \leq |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}} + |b-c|\left(x_1 - x_0 - \frac{p|b-c|}{p+1}\right)^{\frac{1}{p}}$. If $a = b$, $b = c$, or $a = c$, then it is trivially true. If $a < b < c$ or $c < b < a$, then by Lemma 6

$$\begin{aligned}|a-c|\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)^{\frac{1}{p}} &= |a-b|\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)^{\frac{1}{p}} + |b-c|\left(x_1 - x_0 - \frac{p|b-c|}{p+1}\right)^{\frac{1}{p}} \\ &\leq |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}} + |b-c|\left(x_1 - x_0 - \frac{p|b-c|}{p+1}\right)^{\frac{1}{p}},\end{aligned}$$

as $|a-c| \geq |a-b|, |b-c|$. From lemma 6, we know that in the other cases either $|a-c| < |a-b|$ or $|a-c| < |b-c|$. Assume without loss of generality that $|a-c| < |a-b|$. Also, assume that $|a-c|\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)^{\frac{1}{p}} > |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}}$. Then,

$$\begin{aligned}\frac{|a-b|}{|a-c|} &< \left(\frac{x_1 - x_0 - \frac{p|a-c|}{p+1}}{x_1 - x_0 - \frac{p|a-b|}{p+1}}\right)^{\frac{1}{p}} \\ &< \lim_{p\to 0}\left(\frac{x_1 - x_0 - \frac{p|a-c|}{p+1}}{x_1 - x_0 - \frac{p|a-b|}{p+1}}\right)^{\frac{1}{p}} \\ &= e^{\frac{1}{x_1 - x_0}(|a-b| - |a-c|)}\end{aligned}$$

since by Lemma 7 $\left(\frac{x_1 - x_0 - \frac{p|a-c|}{p+1}}{x_1 - x_0 - \frac{p|a-b|}{p+1}}\right)^{\frac{1}{p}}$ is monotonically decreasing for $0 < p < 1$. This shows that

$$\log(|a-b|) - \log(|a-c|) < \frac{1}{x_1 - x_0}(|a-b| - |a-c|)$$

. By Lemma 8, this means that $|a-b| < |a-c|$. Therefore, by the contrapositive,

$$\begin{aligned}|a-c|\left(x_1 - x_0 - \frac{p|a-c|}{p+1}\right)^{\frac{1}{p}} &\leq |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}} \\ &\leq |a-b|\left(x_1 - x_0 - \frac{p|a-b|}{p+1}\right)^{\frac{1}{p}} + |b-c|\left(x_1 - x_0 - \frac{p|b-c|}{p+1}\right)^{\frac{1}{p}}\end{aligned}$$

and D_p satisfies the triangle inequality and so is a metric. □

5 Computer Vision Efficacy

We analyze how well the integral distance performs in analyzing the MNIST data set consisting of handwritten numbers. We used the k-medoids algorithm to classify subsets of a standardized training data set of 60,000 entries into 20 different clusters, and then used a testing data set on a subset of 10,000 entries to assess the clustering accuracy. Each number can be represented with more than one cluster, as the algorithm is intended to recognize more than one way that a number was drawn.

Each cluster was found through the heuristic PAM algorithm detailed in [4]. All 20 medoids for the clusters were randomly chosen, using the previously defined pseudorandom number generator, and then each medoid

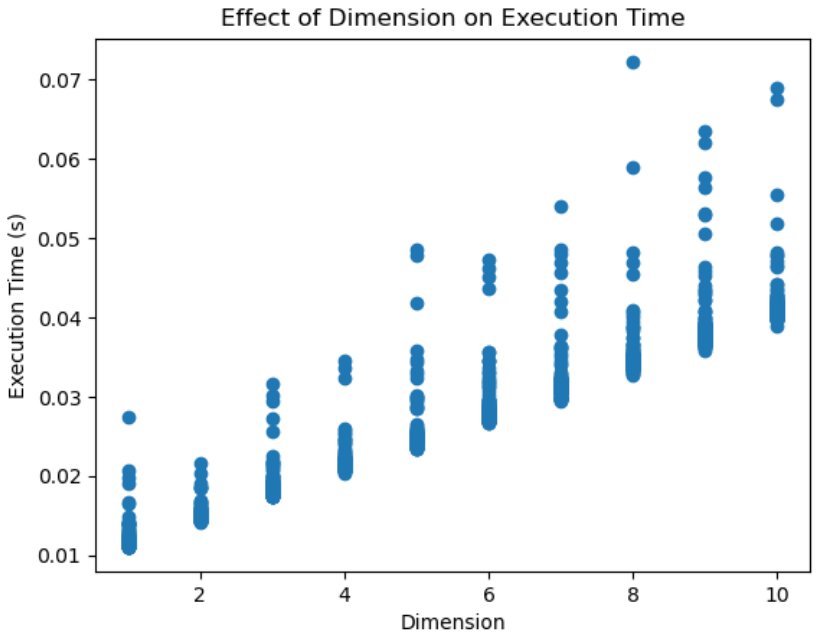
was swapped with each non-medoid in the data set in order to find the minimum of the sum of the integral distance between each medoid and non-medoid. The final 20 medoids were chosen as the 20 points that minimized this distance. Graphs of the results of this approach are presented in the Appendix. Comparisons of the error of this method compared to other methods are also in the Appendix.

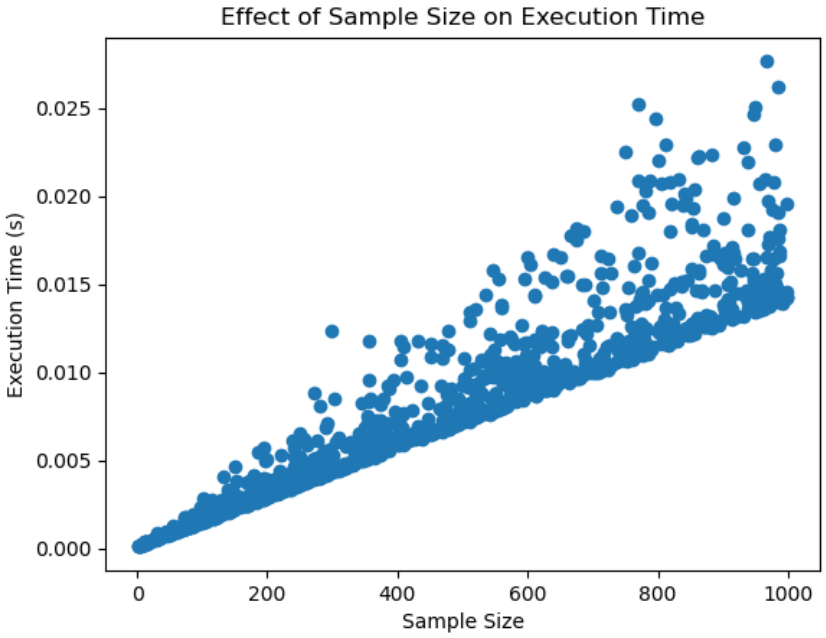
We perform a preliminary analysis of this data set, and we plan to research it more in the future. For this analysis, we ran the k-medoids algorithm 50 times each for the integral distance and the euclidean distance. Each run randomly selected 100 points to train the data, and 25 points to test the classification of the digits. The integral distance marginally outperformed the euclidean distance, with the former correctly classifying 37.6% of the digits, and the latter correctly classifying 34.24% of the digits. The integral distance does have a parameter that can be adjusted to chance how well it performs, and we plan on looking into how this changes the result of the k-medoids algorithm in the future.

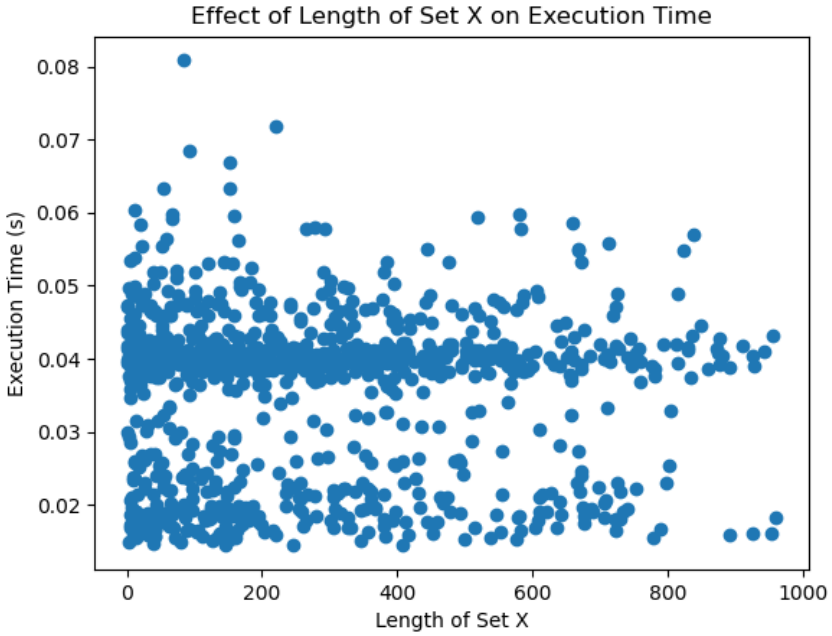
6 Appendix

6.1 Time Complexity Graphs

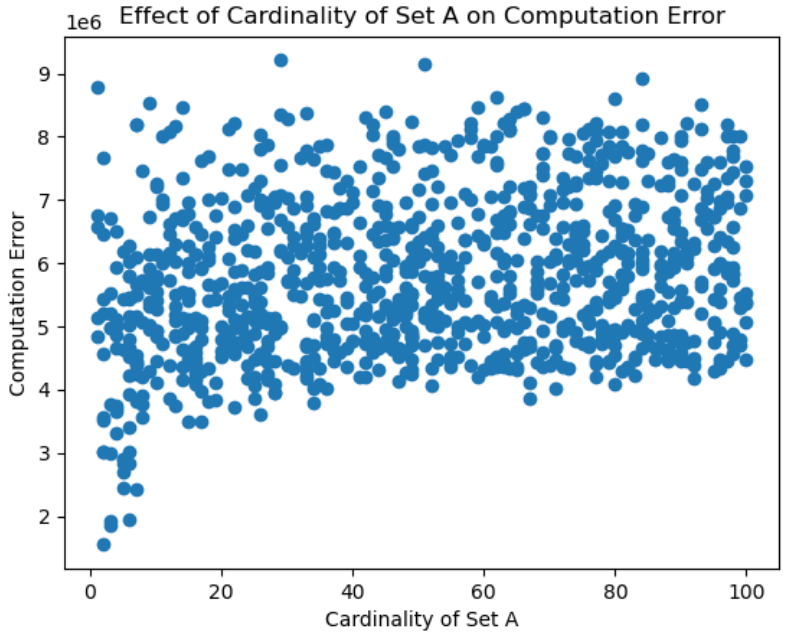
All graphs in 6.1 and 6.2 were generated with 1,000 randomly generated data points, every parameter was held constant except for the parameter listed on the horizontal axis.

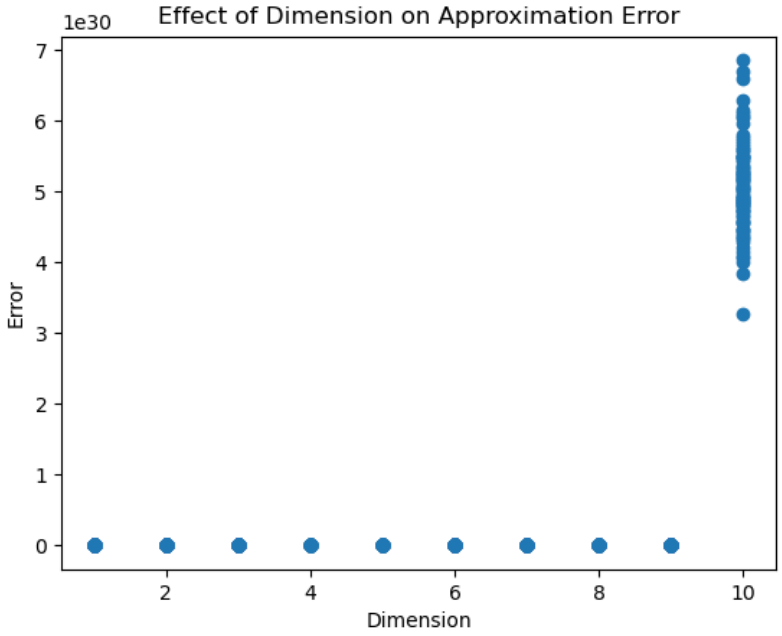

Effect of Dimension on Execution Time
Execution Time (s)
0.07
0.06
0.05
0.04
0.03
0.02
0.01
2
4
6
8
10
Dimension

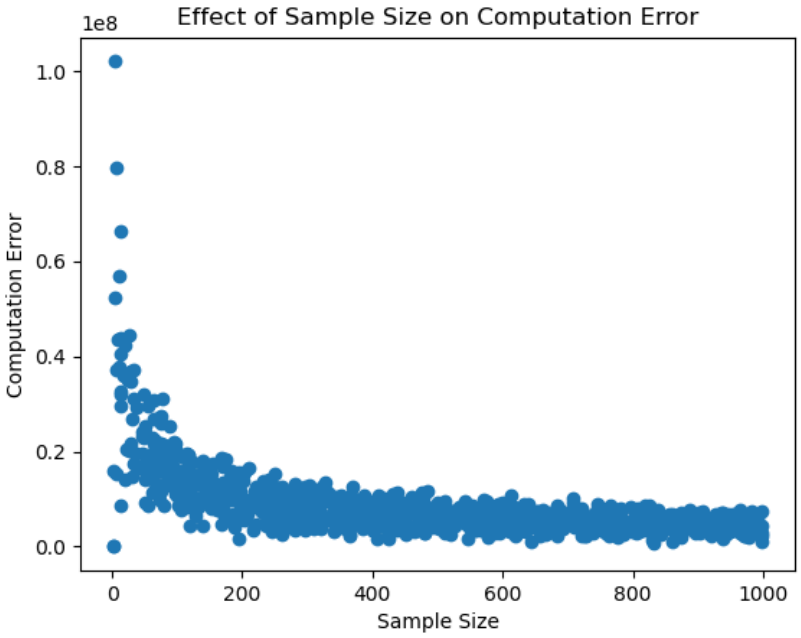

Effect of Sample Size on Execution Time
Execution Time (s)
0.025
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000
Sample Size

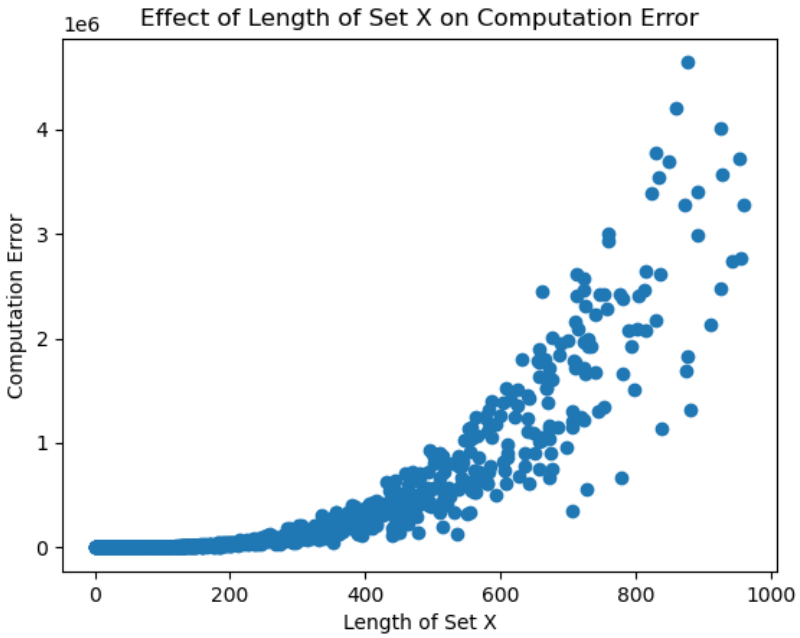


6.2 Error Graphs



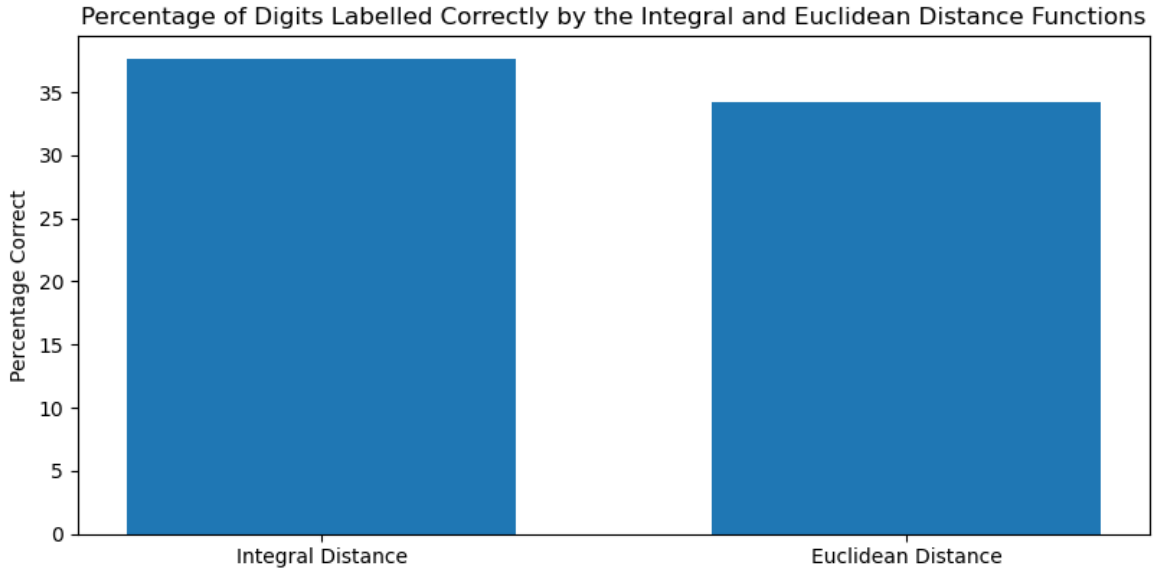

Effect of Dimension on Approximation Error
1e30
7
6
5
4
3
2
1
0
Error
2
4
6
8
10
Dimension


Effect of Sample Size on Computation Error
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Computation Error
0
200
400
600
800
1000
Sample Size



6.3 K-medoids Efficacy

Below is the graph of the efficacy of both distance functions on the MNIST data set, as specificed in Section 5.



6.4 Generated Data

The following is a sample of the results of the Monte-Carlo algorithm computing the integral distance as compared to the Euclidean metric. The sets A and B are the singleton sets being used to compare

the metrics, and the distances from both metrics are printed. We set $p = 1$, the dimension to 1, and $X = [x_0, x_1] = [0, 1000]$ in this case. Alpha and beta are used to look at the bounds for strong equivalence. Alpha and beta are the lower and upper bounds for the integral metric divided by the euclidean metric for the data generated below, respectively. While this doesn't prove anything, the upper bound, beta, is less than $x_1 - x_0$. From extensive testing, beta has always been less than or equal to $(x_1 - x_0)^{\frac{n}{p}}$, where n is the dimension.

```
A: [[906.56758932]]
B: [[9.79422553]]
Euclidean Distance between a and b: 896.7733637855572
Integral Metric between A and B: 500141.65950534324

A: [[196.19749278]]
B: [[613.35387052]]
Euclidean Distance between a and b: 417.15637773805804
Integral Metric between A and B: 330041.93859942263

A: [[582.31092392]]
B: [[547.93622013]]
Euclidean Distance between a and b: 34.374703787599856
Integral Metric between A and B: 33736.111280209065

A: [[933.57678302]]
B: [[137.74831655]]
Euclidean Distance between a and b: 795.8284664744394
Integral Metric between A and B: 485955.0512513258

A: [[244.27753938]]
B: [[441.71639002]]
Euclidean Distance between a and b: 197.43885064719825
Integral Metric between A and B: 175629.55182000058

A: [[858.93638722]]
B: [[553.93315293]]
Euclidean Distance between a and b: 305.00323428612955
Integral Metric between A and B: 258956.7470561114

A: [[779.30507049]]
B: [[638.4099752]]
Euclidean Distance between a and b: 140.89509529388886
Integral Metric between A and B: 130271.77394099499

A: [[120.79494217]]
B: [[436.36621159]]
Euclidean Distance between a and b: 315.5712694179785
Integral Metric between A and B: 267637.00401316746

A: [[880.06865602]]
B: [[937.94514838]]
Euclidean Distance between a and b: 57.87649235430263
Integral Metric between A and B: 56361.3849500228

A: [[779.21683807]]
B: [[979.58247696]]
Euclidean Distance between a and b: 200.36563888979788
```

```
Integral Metric between A and B: 178838.32563905822

Alpha: 557.7124385074184
Beta: 981.4226033382969
```

6.5 Numerical Approximation Program

Displayed below is the python program that randomly generates sets of data that can be plugged into the Monte-Carlo algorithm in order to analyze the integral metric and other metrics. Everything about the set X, the sets A and B, the p-value, the sample size, and the confidence level are customizable, and are randomly generated by default.

```
import numpy as np
import sys

dataAmount = 1000 #Datasets

f = open('dataOutput.txt', 'w') #File to output to
for i in range(dataAmount):
    p = 1
    f.write('p-value: ' + str(p) + '\n') #p-value for the integral
    dimension = np.random.randint(low=1, high=11) #dimension between low and
        high-1
    f.write('Dimension: ' + str(dimension) + '\n')
    highX = 1000 #upper limit of the possible X interval
    low_lim = highX * np.random.random_sample() #lower limit of X interval
        between 0 and highX
    upper_lim = (highX - low_lim) * np.random.random_sample() + low_lim #upper
         limit of X interval between low_lim and highX
    f.write('X interval: [' + str(low_lim) + ', ' + str(upper_lim) + ']\n')
    numPoints = np.random.randint(low=1, high=1001) #sample size between low
        and high-1
    f.write('Sample Size: ' + str(numPoints) + '\n')
    APoints = np.random.randint(low=1, high=1001) #cardinality of the set A;
        between low and high-1
    f.write('Set A: {')
    if (dimension == 1): #If the dimension is 1, don't print out tuples. Only
        print floating points
        for i in range(APoints - 1):
            f.write(str((upper_lim - low_lim) * np.random.random_sample() +
                low_lim) + ', ')
        f.write(str((upper_lim - low_lim) * np.random.random_sample() +
            low_lim))
    else:
        for i in range(APoints): #loop generates a number of points in R^
            dimension equal to the cardinality of the set
            f.write('(')
            for j in range(dimension - 1):
                f.write(str((upper_lim - low_lim) * np.random.random_sample()
                    + low_lim) + ', ')
            f.write(str((upper_lim - low_lim) * np.random.random_sample() +
                low_lim) + ')')
            if(i != APoints - 1): #If this is not the last point to print out,
                output a comma after the ending parenthesis
```

```
                f.write(', ')
    f.write('}\n')

    f.write('Set B: {')
    BPoints = np.random.randint(low=1, high=1001) #cardinality of the set B;
        between low and high-1
    if (dimension == 1): #If the dimension is 1, don't print out tuples. Only
        print floating points
        for i in range(BPoints - 1):
            f.write(str((upper_lim - low_lim) * np.random.random_sample() +
                low_lim) + ', ')
        f.write(str((upper_lim - low_lim) * np.random.random_sample() +
            low_lim))
    else:
        for i in range(BPoints): #loop generates a number of points in R^
            dimension equal to the cardinality of the set
            f.write('(')
            for j in range(dimension - 1):
                f.write(str((upper_lim - low_lim) * np.random.random_sample()
                    + low_lim) + ', ')
            f.write(str((upper_lim - low_lim) * np.random.random_sample() +
                low_lim) + ')')
            if(i != BPoints - 1): #If this is not the last point to print out,
                output a comma after the ending parenthesis
                f.write(', ')
    f.write('}\n')

    confidence = 0.05 #Confidence interval for statistical analysis
    f.write('Confidence Level: ' + str(confidence) + '\n')
    f.write('\n')
f.close()
```

The python program below is the Monte-Carlo Integration algorithm for the integral metric. It also includes computations for the Euclidean metric and the Hausdorff metric. It takes in values from the file generated by the program above, and outputs a file giving the values of the metrics.

```
import numpy as np
import scipy.stats as st
import timeit
import re
from scipy.spatial.distance import directed_hausdorff

betaList = []

#Opens the file, stores the contents, and then closes it
f1 = open('dataOutput.txt', 'r')
contents = f1.readlines()
f1.close()

f2 = open('metricCompare.txt', 'w')

#start = timeit.default_timer()

#Calculates the integral
def integral(x, p):
```

```
    return (abs(x[0] - x[1]) ** p)

#Calculate the infimum using a brute force method
def findInf(x, dimension, A, B):
    #Initializes the infimum values to infinity, so at least something in the
        set will
    #be less than it
    infimumA = np.Infinity
    infimumB = np.Infinity

    #Loops over the A set
    for i in range(len(A)):
        storeA = 0

        #Calculate the euclidean metric for a given dimension
        for j in range(dimension):
            storeA += (x[j] - A[i][j])**2
        storeA = np.sqrt(storeA)

        #Store the lowest value as the infimum
        if (storeA <= infimumA):
            infimumA = storeA

    #Perform the same calculations for the B set
    for i in range(len(B)):
        storeB = 0
        for j in range(dimension):
            storeB += (x[j] - B[i][j])**2
        storeB = np.sqrt(storeB)

        if (storeB <= infimumB):
            infimumB = storeB

    #Return d_A(x) and d_B(x)
    return (infimumA, infimumB)

def integrate(p, dimension, numPoints, X, volume, A, B, confidence):
    #Initialize a vector as the zero vector in R^n, where n is the number of
        dimensions
    x = [0 for i in range(dimension)]
    integralVal = 0.0
    integralSquared = 0.0
    #Loop over all the points that should be calculated
    for i in range(numPoints):
        #Generate a vector in R^n with all values between 0 and 1
        for j in range(dimension):
            x[j] = np.random.uniform(low=X[0], high=X[1])

        #Returns the value of the integral function
        funcval = integral(findInf(x, dimension, A, B), p)

        #Sums up each function value so that it can be used to approximate the
            integral
        #and the integral squared
```

```
        integralVal += funcval
        integralSquared += funcval ** 2


    #Describes the integral
    #print("Integral metric in R^" + str(dimension) + " over [" + str(X[0]) +
        ", " + \
    #str(X[1]) + "]^" + str(dimension) + " with p = " + str(p) + ", A = " +
        str(A) + " and B = " + str(B) + ":")

    #Uses the volume and number of points to calculate the integral value
    integralAverage = integralVal / numPoints
    integralSquaredAverage = integralSquared / numPoints

    solvedIntegral = volume * integralAverage

    euclideanMetric = euclidean(dimension, A, B)
    beta = (solvedIntegral ** (1/p)) / euclideanMetric
    #hausdorff_distance = Hausdorff(A,B)
    #beta = (solvedIntegral ** (1/p)) / hausdorff_distance
    betaList.append(beta)

    #Prints out the integral value
    #print(solvedIntegral**(1/p))
    f2.write('Integral Metric between A and B: ' + str(solvedIntegral ** (1/p)
        ) + '\n\n')

    #print("Standard Error: ")
    #Calculates the standard error of the integral
    stderror = volume * np.sqrt((integralSquaredAverage - (integralAverage **
        2)) / numPoints)

    #print(stderror)
    #Prints a confidence interval for the value of the integral
    zVal = st.norm.ppf(1- (confidence / 2))
    lowLim = solvedIntegral - zVal  * stderror
    upperLim = solvedIntegral + zVal * stderror
    #print(str((1-confidence) * 100) + "% Confidence Interval: (" + str(lowLim
        ) + ", " + str(upperLim) + ")")
    #print("Confidence interval width: " + str(upperLim - lowLim))

def euclidean(dimension, A, B):
    storeMetric = 0

    #Calculate the euclidean metric for a given dimension
    for j in range(dimension):
        storeMetric += (A[0][j] - B[0][j])**2
    storeMetric = np.sqrt(storeMetric)
    f2.write("Euclidean Distance between a and b: " + str(storeMetric) + '\n')
    return storeMetric

#Calculates Hausdorff Distance for A, B
def Hausdorff(A,B):
    hausdorff = max(directed_hausdorff(A,B)[0], directed_hausdorff(B,A)[0])
```

```
    f2.write("Hausdorff Metric bewteen A and B: " + str(hausdorff) + '\n')
    return hausdorff

#Regular expression to check if input from file is a number
reg_ex = re.compile(r"[-+]?\d*\.\d+|\d+")

#Reads in a file with any amount of data, must have the following 8 lines:
#p-value
#dimension
#X interval
#numPoints
#List of numbers for A
#List of numbers for B
#Confidence level
#Empty line
for i in range(len(contents) // 8):
    #Initial declarations of the sets A and B
    A = []
    B = []
    X = []

    p = float(reg_ex.findall(contents[8*i])[0]) #Uses the regular expression
        to get the p-value from the file
    dimension = int(reg_ex.findall(contents[8*i + 1])[0]) #Find the dimension
        from the file
    X.extend([float(i) for i in reg_ex.findall(contents[8*i + 2])]) #Finds the
         X-interval from the file
    volume = (X[1] - X[0]) ** dimension #Calculates the volume of the region
        integrated over
    numPoints = int(reg_ex.findall(contents[8*i+3])[0]) #Grabs the size of the
         sample space
    A.extend([float(i) for i in reg_ex.findall(contents[8*i + 4])]) #Stores
        the points in the set B
    A = np.reshape(A, (len(A) // dimension, dimension)) #Reshapes the set A to
         match the dimension of the space
    B.extend([float(i) for i in reg_ex.findall(contents[8*i + 5])]) #Stores
        the points in the set B
    B = np.reshape(B, (len(B) // dimension, dimension)) #Reshapes the set B to
         match the dimension of the space
    confidence = float(reg_ex.findall(contents[8*i + 6])[0]) #Grabs the
        confidence level
    f2.write('A: ' + str(A) + '\n')
    f2.write('B: ' + str(B) + '\n')
    start = timeit.default_timer() #Starts a timer
    integrate(p, dimension, numPoints, X, volume, A, B, confidence) #
        Integrates using the inputted values
    stop = timeit.default_timer() #Stops the timer
    time = stop - start #Calculates the time needed to calculate the integral
        metric
    #print('')

#Writes to the file to display results of the program
f2.write('Alpha: ' + str(min(betaList)) + '\n')
f2.write('Beta: ' + str(max(betaList)) + '\n')
```

```
betaList = np.array(betaList)
f2.write('Mean: ' + str(betaList.mean()) + '\n')
f2.write('STD: ' + str(betaList.std()))
f2.close()
```

7 Acknowledgements

I would like to thank Dr. Insall for advising me throughout this research and guiding me towards novel ideas to advance this project, and for his help in revising this paper. I would also like to thank Dr. Eric Hanson for his help in verifying the proofs and designing the algorithms in this paper.

References

[1] Charatonik, Wlodzimierz & Insall, Matt. (2006). Metrics on Hyperspaces: A Practical Approach. Topology Proceedings. 30. 129-152.

[2] Weinzierl, Stefan. (2000). Introduction to Monte Carlo Methods. ArXiv High Energy Physics - Phenomenology e-prints. 10.1007/978-0-387-87837-9_1.

[3] Petrusel, Adrian & Rus, Ioan & Serban, Marcel-Adrian. (2013). The role of equivalent metrics in fixed point theory. Topological methods in nonlinear analysis. 41. 85-112.

[4] Schubert, E., & Rousseeuw, P. (2019). Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. ArXiv, abs/1810.05691.

[5] Deza, Michel & Deza, Elena. (2016). Encyclopedia of Distances. 10.1007/978-3-662-52844-0.