DISCRETE ENERGY ANALYSIS ON FRACTAL SETS

CONOR TALIANCICH

Abstract. Principal Component Analysis (PCA) is one of the most commonly-used tools for reducing the dimensionality of large data sets. However, as we will see, PCA is not as effective when applied to data sets which exhibit fractal phenomena. In this paper, we will introduce an alternate tool to supplement PCA called discrete energy analysis, and we will show that it is more effective than PCA at recognizing the true dimensionality of fractal sets.

Contents

1. Introduction

Data analysis is more popular than ever. Many companies routinely analyze data sets with around 1000 variables and a million points. Trying to analyze these data sets directly would be cumbersome and inefficient, so people have come up with various methods to make it easier. One method is to reduce the "effective" dimension of the data set and see if there is a lower-dimensional plane on which many of data points lie together. If we could find such a plane, then analyzing the data set would be much easier since we could restrict our focus to that plane and analyzing its properties.

A popular tool in data science to find the effective dimension of data sets is Principal Component Analysis, which will henceforth be referred to as PCA. PCA is not perfect, however, and sometimes fails to capture the true dimensionality of certain data sets. One class of sets PCA struggles with in particular are sets which exhibit fractal phenomena. This is an issue because fractal phenomena can be observed in various data sets.

Consider the following example. Suppose there is a store with a time series, which is a series of data points collected in chronological order, from the past 20 years detailing sales of their products. Now, suppose that it is observed that each year, the store receives the most sales in January, June, and December. Suppose further that the sales peaked in the second week of each of these respective months, and in those weeks, the sales peaked on Monday, Thursday, and Saturday. Finally, suppose that on those days, the majority the sales happened around the afternoon. This data set exhibits fractal phenomena and is in fact quite similar to a Cantor set.

In order to demonstrate how PCA fails to capture the true dimensionality of these types of sets, we will define discrete Cantor sets, and then we will show through experimentation in Python that PCA has trouble distinguishing these sets from scaled integer lattices. Then, we will define discrete energy analysis and introduce an alternate notion of dimension. Finally, we will use discrete energy analysis to compute the dimension of general discrete Cantor sets and back up our calculations with more experimentation in Python.

2. Principal Component Analysis on Discrete Cantor Sets

2.1. **Principal Component Analysis.**

Let X be a data set with variables $\{v_1, \dots, v_k\}$, where $k \geq 2$. Roughly speaking, given this data set and a positive integer n such that $n \leq k$, PCA identifies the n variables in the data set that account for the most variance in the set and projects the data set onto a plane consisting only of these variables. In this way, PCA identifies the effective dimension of the set by identifying which variables contain the most information about the data set. Now, we will describe how PCA works as follows [1]. A Python implementation of PCA will be included afterwards.

First, for each variable v_i, compute the mean v_i^m and standard deviation z_i of the variable over the set. Then, for each $x \in X$, define x' such that

$$x_i' = \frac{x_i - v_i^m}{z_i},$$

and let X' be the collection of these new points. This standardizes the scale of the variables so that we can accurately compare the variances of different variables.

Next, compute the covariance matrix C of X'. Roughly speaking, this matrix contains information about the correlations between all pairs of variabales in the data set. After that, compute the eigenvalues $\{\lambda_1, \dots, \lambda_k\}$ of C with their respective eigenvalues $\{a_1, \dots, a_k\}$, and sort the eigenvalues from highest to lowest. Each eigenvector points in the direction of one of the variable axes in the data set, and the values of the respective eigenvalues tell us how much the variable accounts for the variance in the data set; the larger the eigenvalue, the more variance that the associated variable accounts for.

Now, take the n variables associated with the n largest eigenvalues, and without loss of generality, suppose they are $v_1, \dots, v_n$ where a_i is the eigenvector pointing in the direction of v_i. These are the principal components, and they will be the variables in the lower-dimensional space we project our data set onto. Finally, compute the product

$$X_{reduced} = [a_1\, a_2\, \cdots\, a_n]^T X'^T.$$

$X_{reduced}$ contains the projection of all the points in the data set onto the plane generated by the n principal components, so now we can analyze this lower-dimensional set instead of the original data set.

However, depending on the choice of n, this projection may retain a lot of information or little information from the original data set. Picking larger n with produce a set that retains more information from the original data set, but the trade-off is that picking smaller n makes the resulting set lower-dimensional and therefore easier to analyze.

Listing 1. Python 3.9.6 Implementation of PCA

```python
import numpy as np
import matplotlib.pyplot as plt

#Parameters: X - 2d array representing a data set
#            n - number of principal components to be selected from X
#Output: graph of the projection of X onto plane generated by its
#        first n principal components
def PCA(X, n):
    #Subtract mean of each variable from all points in the data set
    X_mean = X - np.mean(X , axis = 0)

    #Compute covariance matrix of standardized data set
    cov_mat = np.cov(X_mean , rowvar = False)

    #Compute eigenvalues and eigenvectors of covariance matrix
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    #Sort eigenvalues in descending order along with their corresponding eigenvectors
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    #Take first n eigenvectors
    eigenvector_subset = sorted_eigenvectors[:,0:n]

    #Project standardized data onto plane generated by first n principal components
    X_reduced = np.dot(eigenvector_subset.transpose() , X_mean.transpose() ).transpose()

    #Plot the result
    plt.scatter(X_reduced[:,0],X_reduced[:,1])
plt.show()
```

2.2. **Important Set Constructions.** One of the types of sets experimented on in this paper are lattices in the unit cube $[0,1]^d$. For our purposes, when we say a "lattice with n^d points," we will be referring to the set $\{0, \frac{1}{n}, \ldots, \frac{n-1}{n}, 1\}^d$, which is an evenly spaced grid of points in the unit cube.

The fractal sets we will focus on are "discrete Cantor sets" and direct products of these sets, which we will refer to as discrete Cantor set products. We call these sets discrete Cantor sets because their construction is nearly identical to that of Cantor sets, with the only difference being which points are added to the sets.

The construction of these sets proceeds as follows. Start with the interval $[0,1]$ and positive integers $m, n \in \mathbb{Z}^+$ s.t. $m < n$. Divide $[0,1]$ into n subintervals of equal length, and choose m of those intervals. Let $C^1_{m,n}$ be the set of endpoints of the intervals chosen. At the k-th step, split each of the remaining intervals into n equal subintervals, choose m of those intervals, and let $C^k_{m,n}$ be the set of endpoints of the intervals chosen. At each step, we pick the same m subintervals from the remaining intervals. Since the m subintervals we choose to keep are arbitrary, the set $C^k_{m,n}$ is not unique and merely denotes one set satisfying these conditions.

An an example, consider the sets $C^k_{2,4}$ where we keep the first and third intervals at every step. After the first step, we have $C^1_{2,4} = \{0, 1/4, 1/2, 3/4\}$. Then, after the next step, we would have $C^2_{2,4} = \{0, 1/16, 1/8/, 3/16, 1/2, 9/16, 5/8, 11/16\}$, and so on. It is apparent that $|C^k_{2,4}| = 2^{k+1}$, and in general, $|C^k_{m,n}| = 2m^k$.

Attached below are the implementations of these set constructions in Python.

Listing 2. Python 3.9.6 Implementation of Set Constructions

```
import numpy as np

#Function to create lattice in [0,1]^2
#Parameters: n - positive integer
#Output: a lattice in [0,1]^2 with n^2 equally spaced points
def LatticeSet(n):
    X=np.linspace(0,1,n)
    Y=np.linspace(0,1,n)
    return np.transpose([np.tile(X, len(Y)), np.repeat(Y, len(X))])

#Function to create C_{2,3}^n
#Parameters: n - positive integer
#Output: the set C_{2,3}^n
def cantor(n):
    return [0.] + cant(0., 1., n) + [1.]

#Helper function for cantor that takes a subinterval [x,y] and
#splits it into 2 subintervals
#Parameters: x - left endpoint of interval
#            y - right endpoint of interval
#            n - number of steps remaining in set construction
#Output: the set obtained after repeating n steps of the C_{2,3} construction
#        on [x,y]
def cant(x, y, n):
    if n == 0:
        return [] #Stop construction after the n-th step
    #Add endpoints of 2 new subintervals
    new_pts = [2.*x/3. + y/3., x/3. + 2.*y/3.]
    #Apply construction to the subintervals
    return cant(x, new_pts[0], n-1) + new_pts + cant(new_pts[1], y, n-1)

#Functions to create C_{2,4}^n
#Parameters: n - positive integer
#Output: the set C_{2,4}^n
def cantor_alt(n):
    can = [0,1]
    #Runs n steps of the C_{2,4} construction
    for i in range(n):
        temp = can.copy()
        can = []
        #Iterates over all subintervals in the set and splits them into 2 subintervals
        for j in range(2**i):
            can += cantor_alt_helper([temp[2*j],temp[2*j+1]],i+1)
    return can

#Helper function for cantor_alt that takes an interval [x,y] and
#splits it into 2 subintervals
#Parameters: small_set - interval to be split into 2 subintervals
#            i - positive integer
#Output: the 2 subintervals resulting from small_set
def cantor_alt_helper(small_set,i):
    return [small_set[0], small_set[0]+1/(4**i), small_set[0]+2/(4**i), small_set[0]+3/(4**i)]
```

2.3. **Experimental Results.** As discussed above, PCA is supposed to recognize the "true" dimension of a data set. For many sets, PCA can do this effectively. One such example is a lattice in $[0,1]^d$ for some $d \geq 1$. As we can see from Figure 1, PCA is able to correctly recognize that the lattice is 2-dimensional. However, when we run PCA on the direct product of two or three discrete Cantor sets as is done in Figure 2, Figure 3, and Figure 4, we see that PCA recognizes these sets as scaled down 2-dimensional lattices.

This tells us that PCA has trouble distinguishing these sets from lattices; in other words, PCA "sees" these discrete Cantor set products as 2-dimensional. This is a problem for PCA because the discrete Cantor set products are much more sparse than a 2-dimensional lattice, so treating them as 2-dimensional sets does not capture their true dimensionality. As we will see in the next section, discrete energy analysis outdoes PCA in this regard.

3. Discrete Energy Analysis on Discrete Cantor Sets

3.1. **Discrete Energy Analysis.** Let $\{A_n\}_{n\geq 1} \subseteq [0,1]^d$ (with $d \geq 1$) be a family of finite sets such that $|A_n| = \gamma(n)$ is an increasing function and $A_n \subseteq A_{n+1}$ for all $n \geq 1$. For $s \in [0,d]$, we define the discrete s-energy of A_n to be

$$I_s(A_n) = \gamma(n)^{-2} \sum_{a \neq a'} ||a - a'||^{-s}.$$

We say $\{A_n\}_{n\geq 1}$ is s-adaptable if there exists a constant C such that $I_s(A_n) \leq C$ for all n. If $\{I_s(A_n)\}_{n\geq 1}$ is convergent, then $\{I_s(A_n)\}_{n\geq 1}$ is s-adaptable. The converse may not necessarily be true, however. This fact about s-adaptability is important to keep in mind since it will be used in Section 3.2. Finally, we define

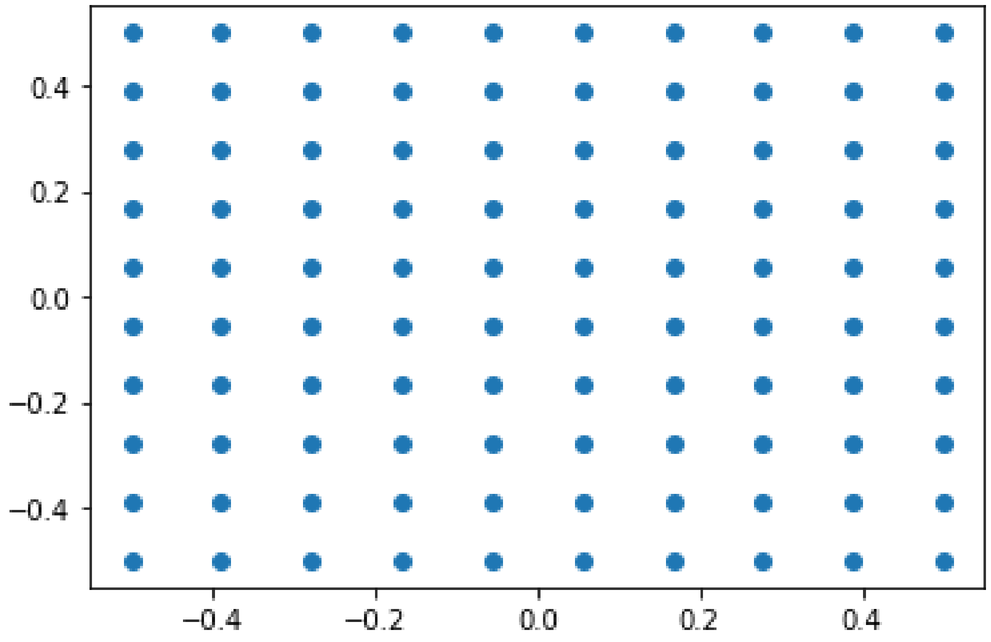

FIGURE 1. PCA with 2 components on 2-dimensional lattice with 10^2 points.

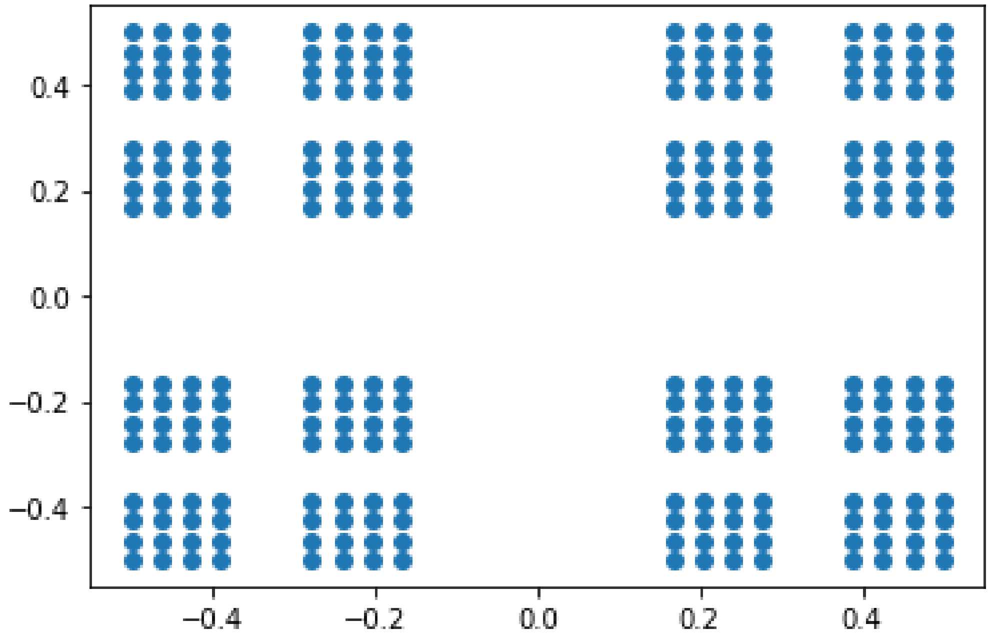

FIGURE 2. PCA with 2 components on $C^3_{2,3} \times C^3_{2,3}$.

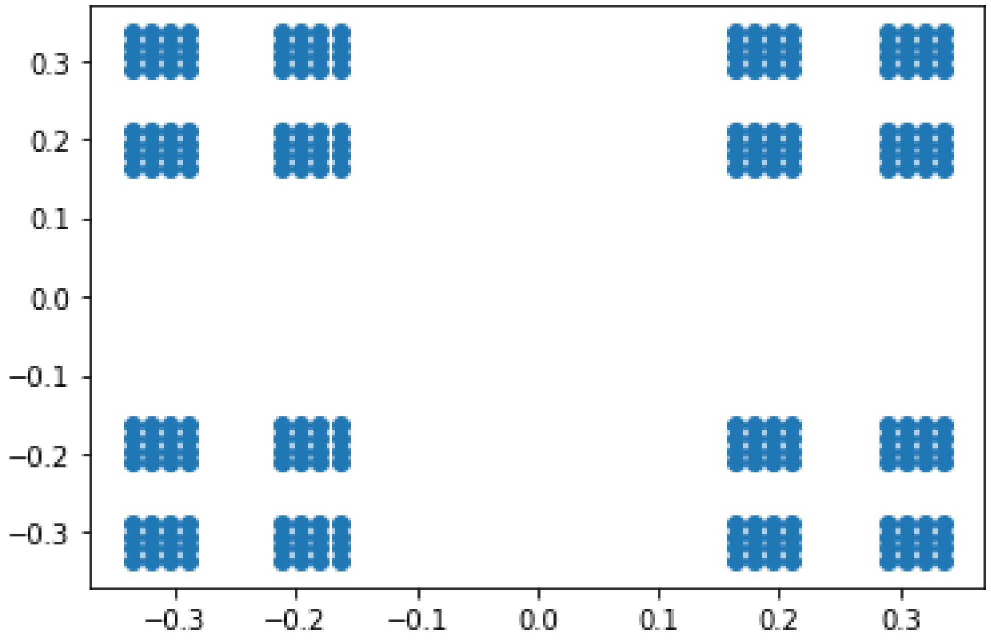

FIGURE 3. PCA with 2 components on $C^3_{2,4} \times C^3_{2,4}$.

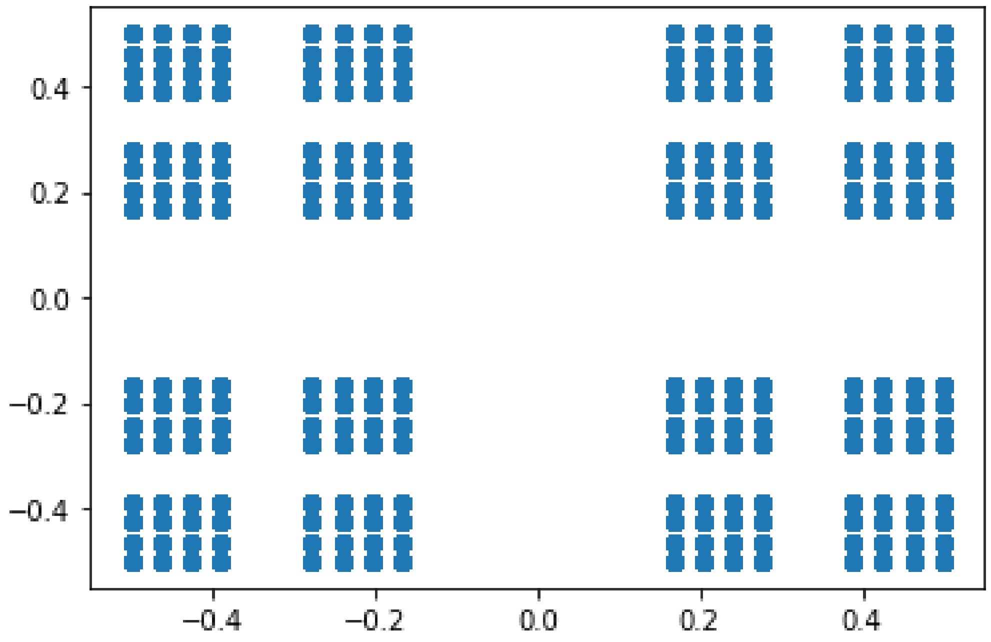

FIGURE 4. PCA with 2 components on $C^3_{2,4} \times C^3_{2,3} \times C^3_{2,3}$.

the critical value $s_{critical}$ for s to be

$$s_{critical} = \sup\{s \in [0, d] : \{A_n\}_{n\geq 1} \text{ is } s\text{-adaptable}\}.$$

The critical value for s is especially important because it gives us a notion of dimension for $\{A_n\}_{n\geq 1}$, which is the discrete version of Hausdorff dimension of $\{A_n\}_{n\geq 1}$ [2]. In fact, this is the notion of dimension we will use below for discrete Cantor set products in lieu of PCA. This method will be henceforth referred to as discrete energy analysis. Attached below is a Python implementation of the discrete energy function in one dimension and multiple dimensions.

Listing 3. Python 3.9.6 Implementation of Discrete Energy Function

```
import numpy as np
import matplotlib.pyplot as plt

#Define the distance function
#Parameters: two arrays of length n representing vectors in R^n
#Output: the norm of the difference between the vectors in R^n
def dist(a,b):
    norm = 0
    for i in range(len(a)):
        norm += (a[i]-b[i])**2
    return norm**.5

#Define the discrete energy function
#Parameters: a discrete set P with dimension d,  an integer s in [0,d],
#           and N = |P|
#Output: the discrete s-energy of P
def DiscreteEnergy(P,s,N):
    c=0
    for i in range(N):
        for j in range(N):
            #Sums over all points p,p' in P such that p != p'
            if i<j:
                c=c+(dist(P[i],P[j]))**(-s)
    return c/N**2

#alternative for efficiency improvements
def DiscreteEnergyAlt(P,s,N):
    c=0
    for i in range(N-1):
        # P[i+1:] is the list of points after P[i]
        # Then, from each point in this list, we subtract P[i]
        diff = P[i+1:] - P[i]
        # Then we compute the norm of each one of those points.
        # Each point is a row of our array P, so we pass
        # the argument axis = 1 to indicate that.
        norms = np.linalg.norm(diff, axis=1)
        # Then we take -s'th power of those numbers and sum them
        c += np.sum(np.power(norms, -s))
    return c/N**2

#One dimensional discrete energy function
#Parameters: a discrete set P with dimension 1,  an integer s in [0,1],
#           and N = |P|
#Output: the discrete s-energy of P
def DiscreteEnergyOneDim(P,s,N):
    c=0
    for i in range(N):
        for j in range(N):
            if i<j:
                #Sums over all points p,p' in P such that p != p'
                c=c+(abs(P[i]-P[j]))**(-s)
    return c/N**2

# essentially identical to DiscreteEnergyAlt
# except the input P is a 1-d array.
# (casting P to a trivial 2-d array and using
# DiscreteEnergyAlt results in slightly worse
# runtime, hence the new definition)
def DiscreteEnergyOneDimAlt(P,s,N):
    P = np.array(P) # making sure we are using numpy array
    c=0
    for i in range(N-1):
        diff = P[i+1:] - P[i]
        c += np.sum(np.abs(diff)**(-s))
    return c/N**2
```

3.2. **Theoretical Results.** Now that we have a new notion of dimension for discrete sets, we can reanalyze the dimension of the sets from Section 2.3. The first thing we will do is compute the dimension of discrete Cantor set products as follows.

Theorem 3.1. *Let $d \in \mathbb{Z}^+$. Let $C^k_{m_1,n_1}, \dots, C^k_{m_d,n_d}$ be discrete Cantor sets. Set $A_k = \prod_{i=1}^d C^k_{m_i,n_i}$. Then*

$$s_{critical} = \frac{\ln(m_1)}{\ln(n_1)} + \cdots + \frac{\ln(m_d)}{\ln(n_d)}.$$

Proof. We will proceed by approximating the sum $|A_k|^{-2}\sum_{a\neq a'} ||a-a'||^{-s}$ by using sets centered at each $a' \in A_k$ such that for each point a in a given set, $|a_i - a'_i| \approx n_i^{-j_i}$ for all i, where $j_1, \dots, j_d$ all range from 1 to k. In this case, $|a_i - a'_i| \approx n_i{}^{-j_i}$ means that $n_i{}^{-j_i-1} < |a_i - a'_i| \leq n_i{}^{-j_i}$. We do not need to consider side lengths with dimensions $n_i^{-j_i}$ for $j_i > k$ because by construction, the i-th discrete Cantor set $C^k_{m_i,n_i}$ does not have any distinct points whose distance is less than n_i^{-k}. Approximating the sum this way gives us

$$|A_k|^{-2}\sum_{a\neq a'} ||a-a'||^{-s} \approx |A_k|^{-2} \sum_{j_1,\dots,j_d=1}^{k} \sum_{\substack{a\neq a' \\ |a_1-a'_1|\approx n_1^{-j_1} \\ \vdots \\ |a_d-a'_d|\approx n_d^{-j_d}}} ||a-a'||^{-s}. \tag{3.1}$$

Next, note that for any $a \neq a'$,

$$||a-a'||^{-s} \approx (|a_1-a'_1| + \cdots + |a_d - a'_d|)^{-s} \leq \max_{1\leq i\leq n}\{|a_i - a'_i|\}^{-s}. \tag{3.2}$$

Putting (4.1) and (4.2) together, we get

$$|A_k|^{-2} \sum_{j_1,\ldots,j_d=1}^{k} \sum_{\substack{a \neq a' \\ |a_1 - a_1'| \approx n_1^{-j_1} \\ \vdots \\ |a_d - a_d'| \approx n_d^{-j_d}}} ||a - a'||^{-s} \approx |A_k|^{-2} \sum_{j_1,\ldots,j_d=1}^{k} \sum_{\substack{a \neq a' \\ |a_1 - a_1'| \approx n_1^{-j_1} \\ \vdots \\ |a_d - a_d'| \approx n_d^{-j_d}}} \max_{1 \leq i \leq n} \{|a_i - a_i'|\}^{-s} \tag{3.3}$$

$$= |A_k|^{-2} \sum_{j_1,\ldots,j_d=1}^{k} \sum_{\substack{a \neq a' \\ |a_1 - a_1'| \approx n_1^{-j_1} \\ \vdots \\ |a_d - a_d'| \approx n_d^{-j_d}}} \max_{1 \leq i \leq n} \{n_i^{-j_i}\}^{-s}. \tag{3.4}$$

Temporarily fix $a' \in A_k$. For any i, the number of elements $a_i \in C^k_{m_i,n_i}$ such that $|a_i - a_i'| \approx n_i^{-j_i}$ is approximately $\frac{|C^k_{m_i,n_i}|}{m_i^{j_i}}$. Therefore, it follows that the number of $a \in A_k$ such that $|a_i - a_i'| \approx n_i^{-j_i}$ for all i is

$$\frac{|C^k_{m_1,n_1}|}{m_1^{j_1}} \cdots \frac{|C^k_{m_d,n_d}|}{m_d^{j_d}} = \frac{|A_k|}{m_1^{j_1} \cdots m_d^{j_d}}.$$

Finally, since there are $|A_k|$ possible choices for a', we can see that

$$|A_k|^{-2} \sum_{j_1,\ldots,j_d=1}^{k} \sum_{\substack{a \neq a' \\ +|a_1 - a_1'| \approx n_1^{-j_1} \\ \vdots \\ |a_d - a_d'| \approx n_d^{-j_d}}} \max_{1 \leq i \leq n} \{n_i^{-j_i}\}^{-s} = |A_k|^{-2} \sum_{j_1,\ldots,j_d=1}^{k} \max_{1 \leq i \leq n} \{n_i^{-j_i}\}^{-s} \sum_{\substack{a \neq a' \\ +|a_1 - a_1'| \approx n_1^{-j_1} \\ \vdots \\ |a_d - a_d'| \approx n_d^{-j_d}}} 1 \tag{3.5}$$

$$\approx |A_k|^{-2} |A_k| |A_k| \sum_{j_1,\ldots,j_d=1}^{k} m_1^{-j_1} \cdots m_d^{-j_d} \max_{1 \leq i \leq n} \{n_i^{-j_i}\}^{-s} \tag{3.6}$$

$$= \sum_{j_1,\ldots,j_d=1}^{k} m_1^{-j_1} \cdots m_d^{-j_d} \max_{1 \leq i \leq n} \{n_i^{-j_i}\}^{-s} \tag{3.7}$$

Now, consider the case where $\max_{1 \leq i \leq n} \{n_i^{-j_i}\} = n_1^{-j_1}$. This would mean that for all i,

$$n_1^{-j_1} \geq n_i^{-j_i} \implies n_1^{j_1} \leq n_i^{j_i} \implies j_1 \frac{\ln(n_1)}{\ln(n_i)} \leq j_i. \tag{3.8}$$

Note that the inequalities in (4.5) also hold for $n_i^{-j_i}$ when it is maximal. Therefore, we can further split the inner sum in (4.4) by considering the different cases in which each $n_i^{-j_i}$ is maximal to get

$$\sum_{j_1,\ldots,j_d=1}^{k} m_1^{-j_1}\cdots m_d^{-j_d} \max_{1\le i\le n}\{n_i^{-j_i}\}^{-s} = \sum_{j_1\frac{\ln(n_1)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_1^{j_1 s} + \cdots + \sum_{j_d\frac{\ln(n_d)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_d^{j_1 s}.$$

We restrict our focus to the first term of the above sum in order to determine for which s the sum converges. For $2 \le i \le n$, the term $n_i^{-j_i}$ in the sum is a geometric series starting at $j_1\frac{\ln(n_1)}{\ln(n_i)}$, so we can approximate each of these terms by $m_i^{j_1\frac{\ln(n_1)}{\ln(n_i)}}$. This gives us that

$$\sum_{j_1\frac{\ln(n_1)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_1^{j_1 s} \approx \sum_{j_1=1}^{n} m_1^{-j_1}\cdots m_d^{-j_1\frac{\ln(n_1)}{\ln(n_d)}} n_1^{j_1 s} = \sum_{j_1=1}^{n}\left(m_1^{-1}\cdots m_d^{-\frac{\ln(n_1)}{\ln(n_d)}} n_1^{s}\right)^{j_1}.$$

Now we have a geometric series, so we know that for this series to converge as $n \to \infty$, we must have that

$$m_1^{-1}\cdots m_d^{-\frac{\ln(n_1)}{\ln(n_d)}} n_1^{s} < 1 \iff -\left(\ln(m_1)+\ln(m_2)\frac{\ln(n_1)}{\ln(n_2)}+\cdots+\ln(m_d)\frac{\ln(n_1)}{\ln(n_d)}\right)+s\ln(n_1) < 0$$
$$\iff s < \frac{\ln(m_1)}{\ln(n_1)}+\cdots+\frac{\ln(m_d)}{\ln(n_d)}.$$

We can use an identical argument to show that all of the other sums

$$\sum_{j_l\frac{\ln(n_1)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_l^{j_l s}$$

also converge exactly when

$$s < \frac{\ln(m_1)}{\ln(n_1)}+\cdots+\frac{\ln(m_d)}{\ln(n_d)}.$$

Furthermore, since

$$I_s(A_k) = |A_k|^{-2}\sum_{a\ne a'}||a-a'||^{-s} \approx \sum_{j_1\frac{\ln(n_1)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_1^{j_1 s}+\cdots+\sum_{j_d\frac{\ln(n_d)}{\ln(n_i)}\le j_i\le k} m_1^{-j_1}\cdots m_d^{-j_d} n_d^{j_1 s},$$

this implies that $\{I_s(A_k)\}_{k\ge1}$ converges exactly when

$$s < \frac{\ln(m_1)}{\ln(n_1)}+\cdots+\frac{\ln(m_d)}{\ln(n_d)}.$$

Hence, $\{A_k\}_{k\ge1}$ is s-adaptable for exactly these values of s, so by definition, we may conclude that

$$s_{critical} = \frac{\ln(m_1)}{\ln(n_1)}+\cdots+\frac{\ln(m_d)}{\ln(n_d)}$$

as desired. □

With this result, if we revisit the discrete Cantor set products from before, then we get that the dimension of $\{C_{2,3}^k \times C_{2,3}^k\}_{k\ge1}$ is $\frac{\ln(2)}{\ln(3)}+\frac{\ln(2)}{\ln(3)} = 2\frac{\ln(2)}{\ln(3)}$, the dimension of $\{C_{2,4}^k \times C_{2,4}^k\}_{k\ge1}$ is $\frac{\ln(2)}{\ln(4)}+\frac{\ln(2)}{\ln(4)} = 1$ and the dimension of $\{C_{2,4}^k \times C_{2,3}^k \times C_{2,3}^k\}_{k\ge1}$ is

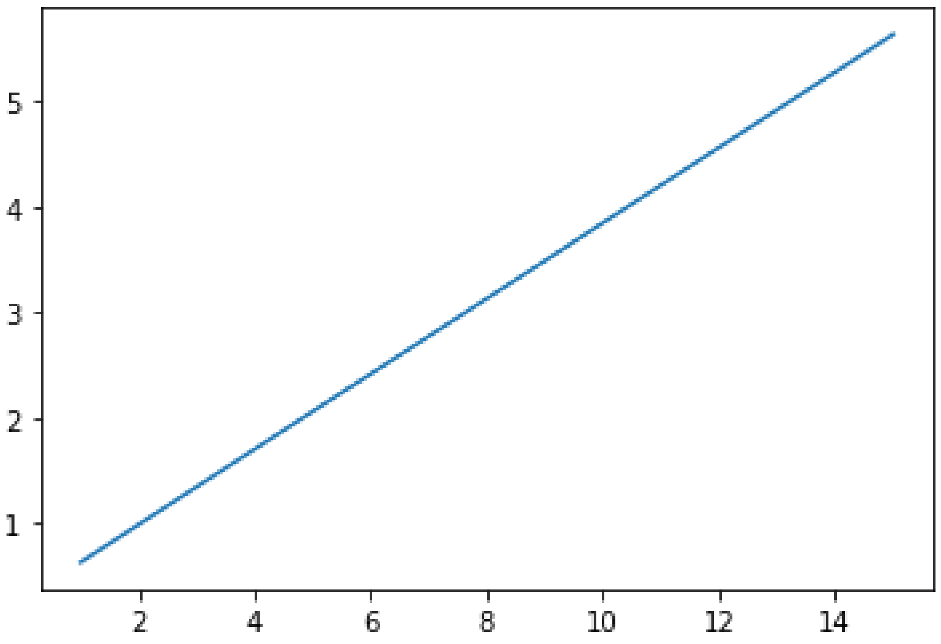

FIGURE 5. Discrete s-energy of $C_{2,4}^{k}$ for $1 \leq k \leq 15$ with $s = \frac{1}{2}$.

$\frac{\ln(2)}{\ln(4)} + \frac{\ln(2)}{\ln(3)} + \frac{\ln(2)}{\ln(3)} = \frac{1}{2} + 2\frac{\ln(2)}{\ln(3)}$, which makes sense given how sparse these sets are. This demonstrates that our alternate notion of dimension is better at capturing the dimensionality of discrete Cantor set products than PCA.

3.3. **Experimental Results.** Now that we have this method computing the dimension of discrete Cantor set products, we can compute the discrete s-energy of the sets in these families to better observe their behavior. For instance, if we compute the discrete s-energy of the discrete Cantor set products from Section 2.3, where for each set, s is the dimension of that set computed by Theorem 3.1, we can see in Figure 5, Figure 6, and Figure 7 that the discrete s-energy of the sets in each case increases slowly. This is what we would expect given that for each family of sets, s is the critical value, and for any $s' < s$, the discrete s'-energy of the sets are all bounded by a single constant.

Meanwhile, if we compute the discrete 2-energy of a family of lattices in $[0,1]^2$, then we can see from Figure 8 that the discrete 2-energy of the lattices increases slowly as well, possibly suggesting that the family of lattices is 2-adaptable. This would make sense intuitively because we would expect the family of lattices to be 2-dimensional, suggesting that this new notion of dimension may align with our general notion of dimension.

4. Future Directions

The next logical step from this point is to experiment more with discrete energy analysis on other fractal sets. While we have shown that discrete energy analysis works better than PCA on discrete Cantor set products, it remains to be seen how

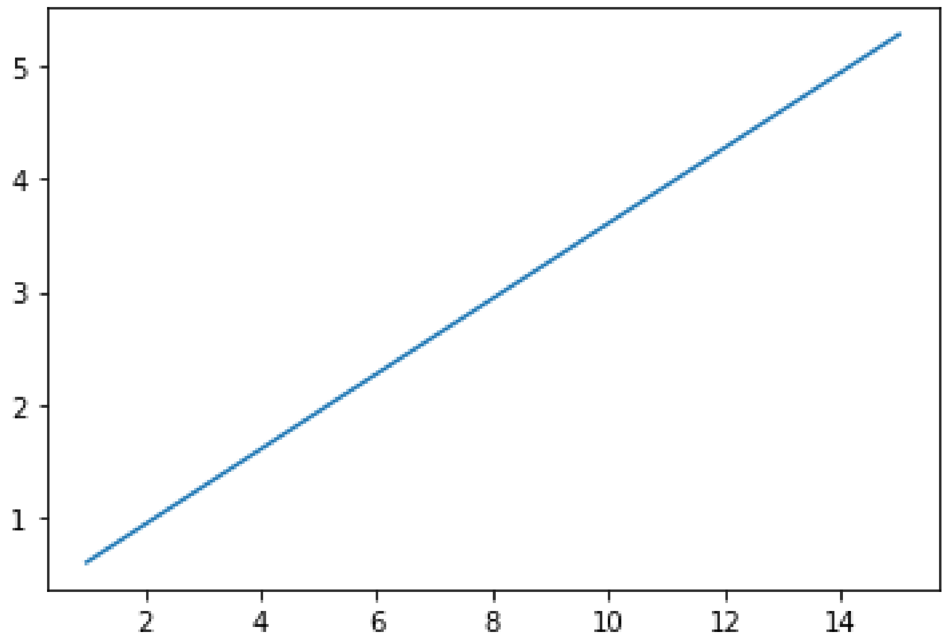

FIGURE 6. Discrete s-energy of $C_{2,3}^k$ for $1 \leq k \leq 15$ with $s = \frac{\ln(2)}{\ln(3)}$.

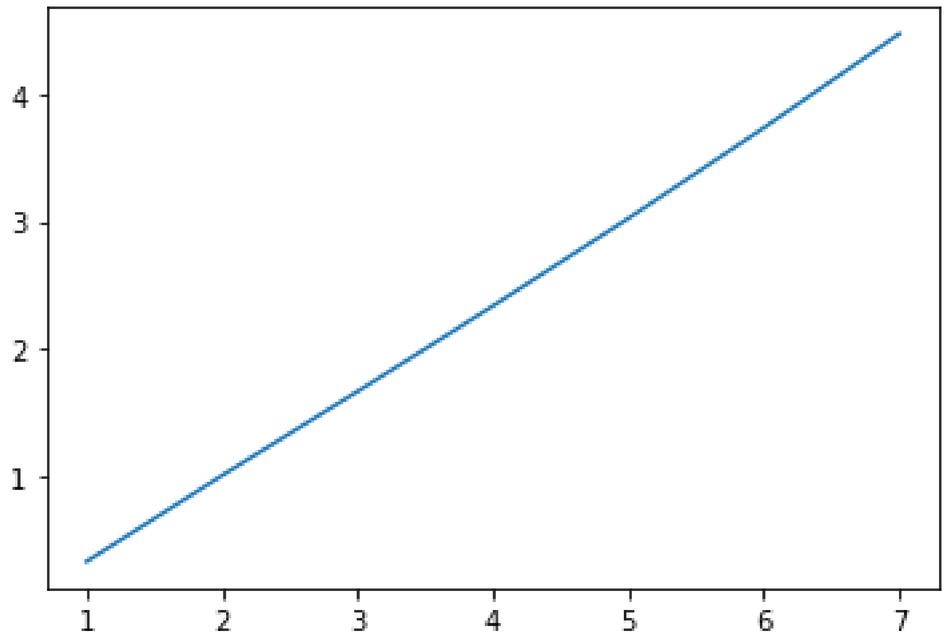

FIGURE 7. Discrete s-energy of $C_{2,4}^k \times C_{2,3}^k$ for $1 \leq k \leq 7$ with $s = \frac{\ln(2)}{\ln(3)} + \frac{1}{2}$.

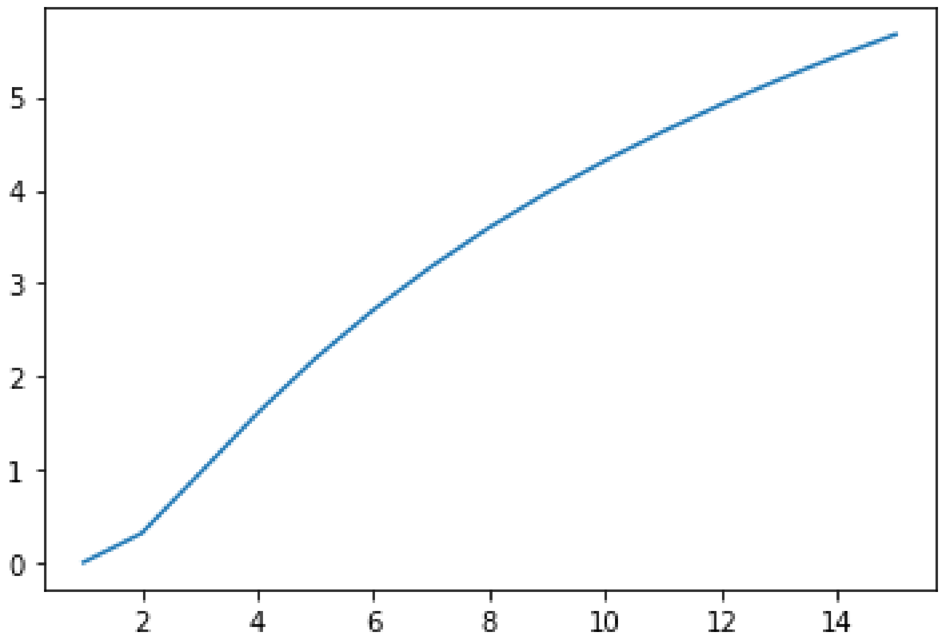

Figure 8. Discrete 2-energy of 2-dimensional lattice with k^2 points for $1 \leq k \leq 15$.

well discrete energy analysis fares when applied to other fractal sets. We may expect discrete energy analysis to work well based on the analysis and experimentation done in this paper, but anything is possible.

It would also be interesting to further explore how the notion of dimension put forward in discrete energy analysis interacts with other notions of dimension. We touched on this briefly in Section 3.3 when looking at the discrete 2-energy of 2-dimensional lattices, but there are many more examples worth trying. For any readers interested in further experimentation, all code used in this paper can be found at https://github.com/ConorT5/ThesisCode2022.

Another thing to consider is why PCA fails when applied to sets exhibiting fractal phenomena. We can see that it does fail when applied to fractal sets of this nature, but we did not explore for what reasons it fails. Figuring out the theory behind why PCA fails would not only give us more insight into the dimensionality properties of the sets on which it does fail, but also will tell us for which sets we should use tools other than PCA to analyze them.

Finally, we reiterate that discrete energy analysis is not being proposed as an alternative to PCA. Despite its shortcomings demonstrated here, PCA is still a valuable tool in data science and as such is widely used. Rather, we propose that discrete energy analysis be used to supplement PCA when PCA falls short.

5. Acknowledgements

I would like to thank my advisor Alex Iosevich for his help with this project, for giving me so many opportunities during my time at the University of Rochester, and especially for being supportive this past year when things were rough. I would like to thank Naomi Jochnowitz, Jonathan Pakianathan, Dan Geba, and Doug Haessig for being amazing professors and fostering my love for the subject. I would like to thank my friends and peers in the University of Rochester Mathematics Department with whom I have spent countless hours laboring over this subject. You made this experience much more fun than I could've ever imagined. Finally, I would like to thank my friends and family outside of the math department. You all were very supportive of me and kept me grounded during my studies.

References

[1] Marc P. Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[2] A. Iosevich, M. Rudnev, and I. Uriarte-Tuero. Theory of dimension for large discrete sets and applications. *Mathematical Modelling of Natural Phenomena*, 9(5):148–169, 2014.