ASYMPTOTIC SPECTRAL PROPERTIES OF THE SCHRÖDINGER OPERATOR WITH THUE-MORSE POTENTIAL

WILLIAM CLARK, RACHAEL KLINE, AND MICHAELA STONE

Abstract. We give background information on the motivations and techniques for answering the open question related to the Thue-Morse sequence posed by Damanik, Embree, and Gorodetski in their 2012 survey of spectral properties of Schrödinger operators arising in the study of quasicrystals. We then report results obtained from our analysis of these spectral properties.

Contents

1. Introduction 1
2. The Thue-Morse Sequence 2
3. Developing the Trace Map 3
4. Matrix Truncation 5
5. Fractal Dimension 6
6. Appendix 14
   Appendix A. Scripts called on in other scripts 14
   Appendix B. Box-counting script 16
   Appendix C. Thickness 19
   Appendix D. Hausdorff (Matrix Truncation) 25
   Appendix E. Hausdorff (Trace Map) 28
References 32

1. Introduction

A crystal is an object which repeats a small pattern. This pattern has a pure period structure in three independent directions and is known as lattice symmetry. X-ray diffraction techniques allow for crystals to be analyzed through their diffraction images. Such images of crystals display the lattice symmetry that is a reciprocal of their underlying crystal structure. In 1982, materials were discovered that displayed a diffraction pattern similar to that of crystals but did not have the characteristics of any lattice symmetry. This discovery introduced a new branch of crystallography that involved materials known as quasicrystals [1]. In 1984, the icosahedral phase of the AlMn alloy was interpreted as a quasiperiodic crystal [6]. As a result, new mathematical works about tilings were developed in attempts to model the geometry involved [1].

In their 2012 survey of spectral properties of Schrödinger operators [3], Damanik, Embree, Gorodetski included the following in their list of open questions: “Study other trace maps (e.g.,
period doubling and Thue-Morse); in particular, find the asymptotics of the Hausdorff dimension of the spectrum as the coupling constant tends to zero or infinity." We use the techniques outlined in this paper to explore the spectral properties of the Thue-Morse Schrödinger operator as the coupling constant tends to zero.

In Section 2 we define the Thue-Morse sequence and its substitution rule. In Section 3 we outline the method of computing the trace map used to approximate the Schrödinger operator with Thue-Morse potential. We then use this trace map to understand the values at which the coupling constant, $V$, are unstable and at which values it is not unstable. The infinite matrix representation of the operator is defined in Section 4 and will be used in parallel with the trace map to estimate the fractal dimension of the spectrum of the operator. We define the notions of fractal dimension which we will be focusing on in Section 5: box-counting dimension, thickness, and Hausdorff dimension. The specific characteristics of the box-counting dimension are discussed in Section 5.2. Similarly we discuss thickness in Section 5.3 and Hausdorff dimension in Section 5.4. Within each of these sections about dimension, we describe the MATLAB code we use to approximate the respective dimension as well as the results. Any MATLAB code referenced in this paper can be found in the appendix preceding the bibliography.

We compute the box-counting dimension, the thickness, and the Hausdorff measure of the spectrum. The results from the box-counting, in Section 5.2.2, reveal that as the coupling constant, $V$, tends to 0, the dimension approaches 1 consistently. Although we are able to numerically estimate thickness, due to the spectrum being sparse the results were not insightful, as discussed in Section 5.3.2. We are able to calculate the Hausdorff measure using two different algorithms. Though we are not able to implement the calculations for Hausdorff dimension, given more time the code presented here could be enhanced to do so. We use the tables generated to approximate the Hausdorff dimension as described in Sections 5.4.2 and 5.4.4. The two different algorithms yield comparable results for a specific coupling constant, $V$.

We would like to thank the Cornell University Summer Math Institute and the NSF. We would also like to thank our advisors May Mei and Andrew Zemke for their support.

2. The Thue-Morse Sequence

The definitions in this section are taken from [2]. One possible construction of quasicrystals is generated using “inflation rules” that utilize a finite alphabet to represent the tiling. A substitution, $\sigma$, assigns a finite string of letters to each letter of the alphabet.

One specific sequence, the Thue-Morse sequence, is comprised of two letters, $a$ and $b$, and is defined by the substitution rule $\sigma(a) = (ab)$ and $\sigma(b) = (ba)$. We denote the sequence as $u$ where $u_n$ is the $n^{th}$ letter in the sequence. Using the substitution, we obtain the sequence $u = abbabaabbaabababba\ldots$, where the sequence is the limit of the substitutions as the length of the sequence goes to infinity. We know the sequence is infinite, because each letter is replaced by two letters with each iteration of the substitution rule. Thus the sequence length doubles with each iteration.

We can also generate the Thue-Morse sequence recursively. To do this we copy the existing $u$, replacing $a$’s with $b$’s and vice versa, and then append the resulting word to the sequence. Bellissard [2] shows that this is equivalent to the substitution method.
3. Developing the Trace Map

The first way in which we can approximate the spectrum of the Schrödinger operator with Thue-Morse potential is through the use of trace maps.

We begin by defining the operator:

**Definition 3.1.** The 1D discrete Schrödinger operator with a Thue-Morse potential defined on the discrete 1-D lattice \( \mathbb{Z} \) where \( V \) is the coupling constant, is

\[
H_V \psi(n) = \psi(n + 1) + V(n)\psi(n) + \psi(n - 1)
\]

where \( V(n) = V(-n - 1) \) and \( V(n) = V \) whenever the \( n^{th} \) letter of \( u \) is \( a \) and \( V(n) = -V \) otherwise (by convention \( u_0 = a \)).

We shall denote the coupling constant by \( V \) for the remainder of this paper.

**Definition 3.2.** Let \( T(n) \) be the transfer matrix:

\[
\begin{bmatrix}
E - V(n) & -1 \\
1 & 0
\end{bmatrix},
\]

where \( E \) is a real parameter.

We can write the Schrödinger equation, \( H\psi = E\psi \), in terms of transfer matrices \( T(n) \) as follows:

\[
\begin{bmatrix}
\psi(n + 1) \\
\psi(n)
\end{bmatrix} = 
\begin{bmatrix}
E - V(n) & -1 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
\psi(n) \\
\psi(n - 1)
\end{bmatrix} = T(n)
\begin{bmatrix}
\psi(n) \\
\psi(n - 1)
\end{bmatrix}.
\]

Then we set \( A = T(0) \) and \( B = T(1) \). \( T(n) = A \) when \( u_n = a \) and \( T(n) = B \) when \( u_n = b \). We are then able to compose the transfer matrix \( M(N) = T(N - 1)T(N - 2)\ldots T(1)T(0) \) from 0 to \( N - 1 \) using the mirror image of \( u \) if \( A \) replaces \( a \) and \( B \) replaces \( b \).

Since the letters appearing at ranks \( 2^{N-1}, \ldots, 2^N - 1 \) are simply those at ranks \( 0, \ldots, 2^{(N-1)} \), substituting \( a \) for \( b \) and vice versa, we set

\[
A(n) = \prod_{i=1}^{2^n} T(2^n - i) \quad A(0) = 1,
\]

\[
B(n) = \prod_{i=1}^{2^n} T(2^n + 1 - i) \quad B(0) = B.
\]

We know by the definition of the sequence that \( B(n) \) is obtained from \( A(n) \) by replacing \( A \) by \( B \) and vice versa, giving us

\[
A(n + 1) = B(n)A(n), \quad B(n + 1) = A(n)B(n).
\]

By the Cayley-Hamilton theorem, any \( 2 \times 2 \) matrix \( M \) with \( \det M = 1 \) satisfies the following equation

\[
M^2 - M.\text{Tr}(M) + 1 = 0.
\]

Then when we set \( x_n = \text{Tr}(A(n)) = \text{Tr}(B(n)) \) (for \( n \geq 1 \)) we get the recursion

\[
x_{n+2} = x_{n+1}x_n^2 - 2x_n^2 + 2, \quad (n \geq 1)
\]
Since $x_{n+1} = x_n^2 - 2$ implies $x_{n+2} = x_{n+1}^2 - 2$, we define a new variable $v_n = x_n^2 - 2 - x_{n+1}$. This variable coincides with $(1/2) \text{Tr}((A(n) - B(n))^2)$. Then

\begin{equation}
\label{eq8}
v_{n+1} = v_n(v_n + 4 - x_n^2), \quad v_1 = 4V^2,
\end{equation}

\begin{equation}
\label{eq9}
x_{n+1} = x_n^2 - 2 - v_n \quad (n \geq 1), \quad x_1 = E^2 - V^2 - 2.
\end{equation}

We can define a mapping $f$ from $\mathbb{R}^2$ to itself by

\begin{equation}
\label{eq10}
f(x, v) = (x^2 - 2 - v, v + 4 - x^2)),
\end{equation}

with the initial conditions $v_1 = 4V^2$, and $x_1 = E^2 - V^2 - 2$.

### 3.1. Implementing the Trace Map.

In order to understand the dynamics of the mapping, we define unstable.

**Definition 3.3.** A point $(x, v)$ is called unstable if there is a neighborhood $\mathcal{U}$ of $(x, v)$ and an integer $n_0$ such that for all $(x_1, v_1)$ in $\mathcal{U}$ and all $N \geq n_0$, the iterated $(x_N, v_N) = f^N(x_1, v_1)$ satisfies $|x_N| > 2$.

**Proposition 3.4.** $E$, energy, values corresponding to unstable points are out of the spectrum of the operator.

We make the transition from the $(x, v)$ plane to $(E, V)$ plane using the conditions stated in [(10)]. Due to the nature of the trace map, the set $D_0 = \{(x, v) : x < -1, v + 4 - x^2 > 4\}$ is unstable. If a point enters the region $D_0$ after some number of iterations, it remains unstable through further iterations of the trace map.

We use `pointtest.m` to test a point $(x, v)$ for several criterion which tells us if the point is unstable or not. We demonstrate this in Figure[1]. If the point is outside of the parabola, $\Sigma_{+,-}$ or $\Sigma_{-,+}$, the point is automatically declared unstable. If the point is on the parabola or within its boundaries below the x-axis, $\Sigma_{-,-}$, the point is characterized as not unstable. When a point is in the $\Sigma_{+,+}$ region, we test whether or not the point enters the region $D_0$.

For the purposes of analysing the spectrum, we must view the $(x, v)$ data as it appears on the $(E, V)$ plane. We use the given initial conditions to implement the $(x, v)$ to $(E, V)$ transformation. Because we are only able to iterate the trace map in the $(x, v)$ plane, we use `linetest.m` to give us cross-section of the data from the $(E, V)$. The code discretizes an $E$ range as specified and tests each point in the discretized data using `pointtest.m`. Any unstable points are denoted as 0 and any not unstable points are denoted as 1. We then output a string of 0’s and 1’s as a cross-section.

The following figures display an $(x, v)$ plot of our data, followed by the corresponding $(E, V)$ plot.
4. Matrix Truncation

Another way we can learn more about the effects of the Schrödinger operator with Thue-Morse potential is by analysing the spectrum via the matrix representation.

**Definition 4.1.** The matrix representation of the Schrödinger operator with Thue-Morse Potential is an infinite tridiagonal matrix with $a_{i,j}$ entries defined as:
\[
\begin{cases}
V(i), & i = j \\
1, & j = i \pm 1 \\
0, & \text{otherwise}
\end{cases}
\]

where \( V(n) \) is the potential at the \( n \)th position in the sequence.

The spectrum of a matrix is defined as the set of its eigenvalues. The spectrum of these finite \( n \times n \) matrices, as \( n \) goes to infinity, approximates the spectrum of the operator.

5. Fractal Dimension

Since the spectrum is a Cantor set with measure zero \([2]\), we wish to analyse the fractal dimension of the spectrum for different coupling constants, \( V \). In particular, we are interested in the behavior of the spectrum as \( V \) approaches 0. We will be using three different notions of fractal dimension to understand the spectra: box-counting dimension, thickness, and Hausdorff dimension.

5.1. How do these dimensions relate? The three dimensions explained above are related in the following way for an arbitrary set \( F \) \([4]\) \([5]\):

\[
\frac{\log 2}{\log \left(2 + \frac{1}{\tau}\right)} \leq \dim_H(F) \leq \dim_B(F):
\]

where \( \tau \) is the thickness, \( \dim_H(F) \) is the Hausdorff dimension, \( \dim_B(F) \) is box-counting dimension.

By calculating both the box-counting dimension and the thickness, we will be able to obtain an upper and lower bound for the Hausdorff dimension.

5.2. Box-Counting. The box-counting dimension of a set \( F \) assumes that there is a solution to the power law \([4]\):

\[
N_\delta(F) \sim c\delta^{-s}
\]

with \( N_\delta \) being the smallest cardinality of a \( \delta \)-cover of \( F \). If we take the logarithm of both sides of the above equation, we get:

\[
\log(N_\delta) \sim \log(c) - s \log(\delta)
\]

which is linear with respect to \( \log \delta \).

Now, we can define the box-counting dimension of a set \( F \) in the following way so that the power law \([12]\) is satisfied \([4]\):

**Definition 5.1.** The box-counting dimension on a set \( F \) is denoted by \( \dim_B(F) \):

\[
\dim_B F = \lim_{\delta \to 0} \frac{\log N_\delta(F)}{-\log \delta}
\]

if the limit exists.
Example 5.1. Let us calculate the box-counting dimension as defined above for the middle-thirds Cantor set.

As we can see from Figure 3, if we let \( \delta = \frac{1}{3} \), then \( N_\delta(F) = 2 \). Now letting \( \delta = \frac{1}{9} \) gives us \( N_\delta(F) = 4 \). By continuing this, we see that if \( \delta = \frac{1}{3^n} \) then \( N_\delta(F) = 2^n \). It follows from Definition 5.1 that the box-counting dimension of the spectrum is \( \dim_B(F) = \frac{\log 2}{\log 3} \).

We can numerically estimate the box-counting dimension of the spectrum by finding the slope of the best fit line of \( \log \delta \) against \( \log N_\delta \) on a log-log graph as observed in (13).

5.2.1. Implementing the Box-Counting Method. To compute the box-counting dimension for a specific \( V \) value, we first must generate a cross-section using pointtest.m and linetest.m as described in Section 3. As before, this cross-section is a vector of 0’s and 1’s; 0’s denote points that are not unstable while 1’s denote points that are unstable.

We compute these vectors for various \( V \) values as specified in masterboxdim.m. In this program, we specify which values of \( V \) to test as well the inputs for linetest.m: the resolution and the number of iterations. For each \( V \) value, the program boxdim.m is called on to calculate an upper and lower estimate for the box-counting dimension of that cross-section. However, if the cross-section is entirely 0’s then the program returns an error stating that there are no points in the spectrum.

To calculate the box-counting dimension of a set, we have to determine the slope of the best-fit line of the plot of \( \log \delta \) against \( \log N_\delta \). For varying \( \delta \) values, boxdim.m finds a covering for all of the 1’s in the cross-section vector to find an estimate for \( N_\delta \). For each given \( \delta \), the program implements the following algorithm to find \( N_\delta \). The \( \delta \) value is translated into an integer number called res, which is the number of points in the cross-section vector that has length \( \delta \) if scaled onto the trace map. Beginning at the far left of the vector, the program searches until the first 1 is found. Then the program goes over to the right by res – 1 points and those points make up a cover of length \( \delta \). After this cover has been found, the program adds one to the \( N_\delta \) count and continues to the right until the next 1 is found or until the whole set is exhausted. Once the whole set has been searched, the number of sets needed to cover all of the 1’s is the calculated \( N_\delta \) number. The best-fit line of \( \log \delta \) versus \( \log N_\delta \) will be the box-counting dimension if all of the points are co-linear. However, we see that this is not the case. If we find the best-fit line for the first half of our points and the second half separately, we obtain two different estimates for the box-counting dimension.

Plotting the upper and lower estimates for the box-counting dimension at multiple \( V \) values, the masterboxdim.m program yields the following results:
5.2.2. Results of Box-Counting Computations. After running `boxdim.m`, the following figure is generated:

![Figure 4](image1.png)

**Figure 4.** This is a log-log plot of $\delta$ versus $N_\delta$ with a coupling constant of 0.3 and with the resolution checking $2^{17}$ points under 10,000 iterations. The results are partitioned into two groups so the data is being fitted with two best-fit lines for the upper and lower estimates on the box-counting dimension.

As we can see from Figure 4, the confidence level of a best fit line across all the data points is misleading. By breaking up the data into two separate sections, we can get a better approximation for the box-counting dimension as we obtain an upper and lower estimate.

We generate the following figure using `masterboxdim.m`:

![Figure 5](image2.png)

**Figure 5.** This graph is a $V$ versus $\text{dim}_B$ plot, where $V$ is the coupling constant and $\text{dim}_B$ is the box-counting dimension of the spectrum cross-section at that $V$. The graph includes both the upper and lower estimates for the box-counting dimension as described above.
We can see that as $V$ approaches 0, the upper estimate is seemingly approaching 1 linearly. Although the lower estimate has linear tendencies, it seems to be more constant. From these results, we conclude with high confidence that the box-counting dimension lives between these two estimates.

5.3. Thickness. Another type of fractal dimension we will consider is the thickness of a Cantor set. Here, we define the terms relevant to computing thickness.

**Definition 5.2.** Let $K \subset \mathbb{R}$. A gap of $K$ is a connected component $\mathbb{R} \setminus K$; a bounded gap is a bounded connected component of $\mathbb{R} \setminus K$. Let $U$ be any bounded gap and $u$ be a boundary point of $U$, so $u \in K$. Let $C$ be the bridge of $K$ at $u$, i.e. the maximal interval in $\mathbb{R}$ such that
- $u$ is a boundary point of $C$,
- $C$ contains no point of a gap $U'$ whose length $\ell(U')$ is at least the length of $U$.

**Definition 5.3.** The thickness of $K$ at $u$ is defined as $	au(K,u) = \ell(C)/\ell(U)$. The thickness of $K$, denoted by $\tau(K)$, is the infimum over these $\tau(K,u)$ for all boundary points $u$ of bounded gaps.

**Example 5.2.** The following diagram demonstrates the computation of the thickness of the middle-thirds Cantor set.

![Diagram of the middle-thirds Cantor set with gaps and bridges]

Figure 6. This illustrates the computation of thickness using on the second iteration of the middle-thirds Cantor set on the closed interval $[0,1]$, see Figure 3. The top dotted interval represents $C_{u_3}$, and the bottom dotted interval represents $C_{u_1}$.

To compute the thickness of this set, first consider the gap $U$ bounded by $u_1$ and $u_2$. We must construct the bridge $(C,u_1)$ as described in Definition 5.2. Note that if we attempt to construct $C$ going to the right of $u_1$, the bridge would contain the gap $U$. This violates the second condition in Definition 5.2. Hence we must construct $C$ over the interval $[0,u_1]$. Then we have that:

$$\tau(K,u_1) = \ell(C)/\ell(u_1) = \frac{1}{\frac{1}{3}} = 1.$$ 

Similarly, if we wish to compute the thickness at the gap with the boundary $u_3$, we must construct $(C,u_3)$ as illustrated and the result is

$$\tau(K,u_3) = \ell(C)/\ell(u_3) = \frac{1}{\frac{3}{5}} = 1.$$ 

Indeed, for all $u_i$ in the middle-thirds cantor set, we will find that $\tau(K,u_i) = 1$. Then to compute $\tau(K)$ we take the infimum of the thicknesses at each $u_i$, giving us

$$\tau(K) = \inf \{ \tau(K,u_i) \} = 1.$$ 

We will be applying this technique to cross-sections of the plots of the data in order to approximate thickness as $V$ approaches 0.
5.3.1. Implementing the Thickness Computations. To compute the thickness for a specific \( V \) value, we first must calculate a cross-section using `pointtest.m` and `linetest.m` as described in Section 3. As before, this cross-section is made up of 0’s and 1’s; 0’s denote points that are unstable while 1’s denote points that are not unstable.

We compute a cross-section for several different \( V \) values. In `masterthickness.m`, we allocate which \( V \) values we want to compute thickness for as well as the appropriate inputs for `linetest.m`. For each \( V \) cross-section, we compute the thickness using `thickness.m`. This code only calculates the thickness if there are both 0’s and 1’s in the cross-section of interest. If the cross-section consists of only 0’s, we know that those points are not in the spectrum. If the cross-section contains all 1’s, then we know the thickness will be infinity. Since both of these computations are irrelevant for our purposes, we focus on the cross-sections containing both gaps and intervals.

To calculate the thickness, we input the cross-section into `gapEval.m`. This program then uses `gapFinder.m` to calculate an \( n \times 3 \) matrix to store the information about the gaps of the cross-section, where \( n \) is the total number of gaps. In the cross-section, gaps are strings of 0’s. After this matrix is created, `gapEval.m` uses it to calculate a bridge for each endpoint \( u_i \) as defined in Definition 5.2. A new \( m \times 3 \) matrix, `thicku`, is created where each row corresponds to an endpoint \( u_i \). We use the third column of this matrix to calculate the \( \tau(K, u_i) \) as defined in Definition 5.3. The following example will demonstrate the processes in the programs.

**Example 5.3.** Suppose we have a cross-section:

\[
(1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1)
\]

Using `gapFinder.m`, we get the matrix with the entries:

<table>
<thead>
<tr>
<th>Beginning of the gap</th>
<th>Point to the right of the end of the gap</th>
<th>Length of the gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>

Using `gapEval.m`, we get a \( 6 \times 3 \) matrix with the following entries:

<table>
<thead>
<tr>
<th>Length of the bridge</th>
<th>Length of the gap</th>
<th>Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

gapEval.m then returns the thickness, the infimum over these \( \tau(K, u_i) \), for this cross-section at a specific \( V \).

We repeat these calculations to obtain various thickness computations for specific \( V \) values. We then use the relation stated in (11) and plot the output to obtain a lower bound for Hausdorff dimension.

5.3.2. Results of Thickness Computations. The following figure is generated using the data from `masterthickness.m`:
Figure 7. A plot of $V$ versus $f(\tau)$ where $f(\tau) = \log 2/\log (2 + (1/\tau))$ which is a lower bound on the Hausdorff dimension.

Compared to the results of the box-counting dimension in Section 5.2.2, the thickness approximation we calculated is not as precise. Figure 7 shows that $f(\tau)$ is generally decreasing as $V$ increases. Though this is consistent with the box-counting dimension results, the range between these two bounds for Hausdorff dimension as described in (11) is distant.

5.4. Hausdorff Dimension. Before the Hausdorff dimension can be defined, the Hausdorff measure of a set must be defined [4].

Definition 5.4. The Hausdorff measure of a set $F$ of dimension $s$, denoted by $\mathcal{H}^s(F)$, is:

\begin{equation}
\mathcal{H}^s(F) = \lim_{\delta \to 0} \left\{ \inf \left\{ \sum_{i=1}^{\infty} |U_i|^s : \{U_i\} \text{ is a } \delta \text{-cover of } F \right\} \right\}.
\end{equation}

With this in mind, we can define the Hausdorff dimension as follows [4]:

Definition 5.5. The Hausdorff dimension of a set $F$, denoted by $\text{dim}_H(F)$, is:

\begin{equation}
\text{dim}_H(F) = \inf \{ s \geq 0 : \mathcal{H}^s(F) = 0 \} = \sup \{ s : \mathcal{H}^s(F) = \infty \}
\end{equation}

Specifically, $\text{dim}_H(F)$ is the value of $s$ in (15) where the Hausdorff measure ‘jumps’ from $\infty$ to 0 [4].
5.4.1. Using the Trace Map to Compute Hausdorff Dimension. We can calculate Hausdorff dimension for a specific $V$ using a cross-section generated with `pointtest.m` and `linetest.m` as described in Section 3. In `masterDorff.m`, we can calculate the Hausdorff measure at a specific $V$ value as defined in Definition 5.4 for several different $s$ values.

We first specify which $V$ value we want to calculate the Hausdorff measure for, the different $s$ values we want to evaluate, and the inputs for `linetest.m`. We calculate the cross-section using a specific resolution, the number of points that are in the cross-section. Using this resolution, we vary the length of the $\delta$-cover to calculate the Hausdorff measure as the $\delta$-cover approaches 0. Now, for each of the different $s$ values previously specified in the code, we call on `deltaDorff.m` to calculate the Hausdorff measure at each of the different $\delta$ values. In `deltaDorff.m`, we create an $n \times 3$ matrix to store the different positions and length of the $\delta$-coverings. Within `deltaDorff.m`, we first analyse where the intervals are located within the cross-section; then, we analyse these intervals to ensure they are at most length $\delta$. We continue by using this $n \times 3$ matrix to calculate the Hausdorff measure. We demonstrate this process in the following example:

Example 5.4. Suppose we have a cross-section:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$  

Suppose we specify $\delta$ to be of length 2. Using `deltaDorff.m`, we get a $4 \times 3$ matrix:

<table>
<thead>
<tr>
<th>Beginning point of the string of 1’s</th>
<th>Point to the right of the string of 1’s</th>
<th>Length of the string of 1’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

We then reevaluate the previous matrix to make sure the $\delta$-cover is at most length $\delta$ and we obtain a $6 \times 3$ matrix:
After obtaining the measures for the varying \( s \) values across several different \( \delta \)-covers, we plot the results. The Hausdorff dimension is interpreted as the \( s \) value at which the Hausdorff measure changes from approaching infinity and begins approaching zero. With both a plot and a table, we can analyse our data to approximate the value of the Hausdorff dimension at a specific \( V \) value.

5.4.2. Results of Trace Map Computations of Hausdorff Dimension. We generate the following table using \texttt{masterDorff.m} at a \( V = 0.03 \):

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 0.00 )</th>
<th>( 0.11 )</th>
<th>( 0.22 )</th>
<th>( 0.33 )</th>
<th>( 0.44 )</th>
<th>( 0.56 )</th>
<th>( 0.67 )</th>
<th>( 0.78 )</th>
<th>( 0.89 )</th>
<th>( 1.00 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>533</td>
<td>267.752</td>
<td>134.560</td>
<td>67.654</td>
<td>34.031</td>
<td>17.127</td>
<td>8.625</td>
<td>4.346</td>
<td>2.191</td>
<td>1.106</td>
</tr>
<tr>
<td>10</td>
<td>533</td>
<td>247.905</td>
<td>115.351</td>
<td>53.697</td>
<td>25.008</td>
<td>11.653</td>
<td>5.433</td>
<td>2.535</td>
<td>1.183</td>
<td>0.553</td>
</tr>
<tr>
<td>11</td>
<td>533</td>
<td>229.529</td>
<td>98.884</td>
<td>42.619</td>
<td>18.378</td>
<td>7.929</td>
<td>3.423</td>
<td>1.478</td>
<td>0.639</td>
<td>0.276</td>
</tr>
<tr>
<td>12</td>
<td>538</td>
<td>214.400</td>
<td>85.468</td>
<td>34.082</td>
<td>13.596</td>
<td>5.426</td>
<td>2.166</td>
<td>0.865</td>
<td>0.346</td>
<td>0.138</td>
</tr>
<tr>
<td>13</td>
<td>566</td>
<td>207.973</td>
<td>76.418</td>
<td>34.082</td>
<td>13.596</td>
<td>5.426</td>
<td>2.166</td>
<td>0.865</td>
<td>0.346</td>
<td>0.138</td>
</tr>
<tr>
<td>14</td>
<td>1132</td>
<td>356.567</td>
<td>112.314</td>
<td>35.378</td>
<td>11.144</td>
<td>3.510</td>
<td>1.106</td>
<td>0.348</td>
<td>0.110</td>
<td>0.035</td>
</tr>
<tr>
<td>15</td>
<td>2264</td>
<td>611.329</td>
<td>165.072</td>
<td>44.573</td>
<td>12.036</td>
<td>3.250</td>
<td>0.878</td>
<td>0.237</td>
<td>0.064</td>
<td>0.017</td>
</tr>
<tr>
<td>16</td>
<td>4528</td>
<td>1048.12</td>
<td>242.612</td>
<td>56.159</td>
<td>12.999</td>
<td>3.009</td>
<td>0.697</td>
<td>0.161</td>
<td>0.037</td>
<td>0.009</td>
</tr>
<tr>
<td>17</td>
<td>9056</td>
<td>1796.98</td>
<td>356.575</td>
<td>70.755</td>
<td>14.040</td>
<td>2.786</td>
<td>0.553</td>
<td>0.110</td>
<td>0.022</td>
<td>0.004</td>
</tr>
<tr>
<td>18</td>
<td>18112</td>
<td>3080.91</td>
<td>524.071</td>
<td>89.146</td>
<td>15.164</td>
<td>2.579</td>
<td>0.439</td>
<td>0.075</td>
<td>0.013</td>
<td>0.002</td>
</tr>
<tr>
<td>19</td>
<td>36224</td>
<td>5282.18</td>
<td>770.246</td>
<td>112.317</td>
<td>16.378</td>
<td>2.388</td>
<td>0.348</td>
<td>0.051</td>
<td>0.007</td>
<td>0.001</td>
</tr>
<tr>
<td>20</td>
<td>72448</td>
<td>9056.23</td>
<td>1132.057</td>
<td>141.511</td>
<td>17.689</td>
<td>2.211</td>
<td>0.276</td>
<td>0.035</td>
<td>0.004</td>
<td>0.001</td>
</tr>
<tr>
<td>21</td>
<td>144896</td>
<td>15526.8</td>
<td>1663.823</td>
<td>178.292</td>
<td>19.105</td>
<td>2.047</td>
<td>0.219</td>
<td>0.024</td>
<td>0.003</td>
<td>0.000</td>
</tr>
<tr>
<td>22</td>
<td>289792</td>
<td>26620.5</td>
<td>2445.377</td>
<td>224.634</td>
<td>20.635</td>
<td>1.896</td>
<td>0.174</td>
<td>0.016</td>
<td>0.001</td>
<td>0.000</td>
</tr>
<tr>
<td>23</td>
<td>579584</td>
<td>45640.5</td>
<td>3594.056</td>
<td>283.021</td>
<td>22.287</td>
<td>1.755</td>
<td>0.138</td>
<td>0.011</td>
<td>0.001</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The table above shows the values of Hausdorff measure as defined in [15]. We approximate the value of the Hausdorff dimension by analysing the limits as \( \delta \) approaches 0 for various \( s \) values; the \( s \) value at which the Hausdorff measure changes from approaching infinity and begins to tend towards 0 is the approximation for the Hausdorff dimension. The table reveals this approximation to be around 0.4.

5.4.3. Using Matrix Truncation to Compute Hausdorff Dimension. In Section 4, we discuss the use of matrix truncation to approximate Hausdorff dimension. In order to make these computations, we use \texttt{potentialMatrix.m} and \texttt{eigendorff.m}.

In \texttt{potentialMatrix.m}, we create the matrix representation for the Schrödinger operator. The diagonal is populated with the potentials from the Thue-Morse sequence, and the sub- and super-diagonals are populated with ones. Finally, the script saves the resulting matrix as a sparse matrix and computes its eigenvalues.
The eigenvalues are then used by `eigenDorff.m` to compute the Hausdorff measure calculations. This script finds suitable $\delta$-covers and uses them to compute the Hausdorff measure of 10 different $s$ values. It outputs a matrix in which each column gives the list of Hausdorff measures for a particular $V$ value using the selected $\delta$-covers.

5.4.4. Results of Matrix Truncation Computations of Hausdorff Dimension. We generate the following table using `eigenDorff.m` at a $V = 0.03$ with a $1000 \times 1000$ matrix:

<table>
<thead>
<tr>
<th>$s$-values</th>
<th>0.00</th>
<th>0.11</th>
<th>0.22</th>
<th>0.33</th>
<th>0.44</th>
<th>0.56</th>
<th>0.67</th>
<th>0.78</th>
<th>0.89</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>9.259</td>
<td>8.572</td>
<td>7.937</td>
<td>7.349</td>
<td>6.804</td>
<td>6.300</td>
<td>5.833</td>
<td>5.400</td>
<td>5.000</td>
</tr>
<tr>
<td>5</td>
<td>130</td>
<td>88.451</td>
<td>60.182</td>
<td>40.947</td>
<td>27.860</td>
<td>18.956</td>
<td>12.898</td>
<td>8.775</td>
<td>5.971</td>
<td>4.063</td>
</tr>
<tr>
<td>6</td>
<td>258</td>
<td>162.530</td>
<td>102.387</td>
<td>64.500</td>
<td>40.632</td>
<td>25.597</td>
<td>16.125</td>
<td>10.158</td>
<td>6.399</td>
<td>4.031</td>
</tr>
<tr>
<td>7</td>
<td>506</td>
<td>295.132</td>
<td>172.140</td>
<td>100.403</td>
<td>58.562</td>
<td>34.157</td>
<td>19.923</td>
<td>11.620</td>
<td>6.778</td>
<td>3.953</td>
</tr>
<tr>
<td>8</td>
<td>769</td>
<td>415.283</td>
<td>224.265</td>
<td>121.110</td>
<td>65.403</td>
<td>35.320</td>
<td>19.074</td>
<td>10.300</td>
<td>5.562</td>
<td>3.004</td>
</tr>
<tr>
<td>9</td>
<td>891</td>
<td>445.500</td>
<td>222.750</td>
<td>111.375</td>
<td>55.688</td>
<td>27.844</td>
<td>13.922</td>
<td>6.961</td>
<td>3.480</td>
<td>1.740</td>
</tr>
<tr>
<td>10</td>
<td>950</td>
<td>439.790</td>
<td>203.595</td>
<td>94.252</td>
<td>43.633</td>
<td>20.199</td>
<td>9.351</td>
<td>4.329</td>
<td>2.004</td>
<td>0.928</td>
</tr>
<tr>
<td>11</td>
<td>976</td>
<td>418.335</td>
<td>179.308</td>
<td>76.855</td>
<td>32.942</td>
<td>14.120</td>
<td>6.052</td>
<td>2.594</td>
<td>1.112</td>
<td>0.477</td>
</tr>
<tr>
<td>12</td>
<td>988</td>
<td>392.088</td>
<td>155.600</td>
<td>61.750</td>
<td>24.506</td>
<td>9.725</td>
<td>3.859</td>
<td>1.532</td>
<td>0.608</td>
<td>0.241</td>
</tr>
<tr>
<td>13</td>
<td>996</td>
<td>365.964</td>
<td>134.467</td>
<td>49.408</td>
<td>18.154</td>
<td>6.670</td>
<td>2.451</td>
<td>0.901</td>
<td>0.331</td>
<td>0.122</td>
</tr>
<tr>
<td>14</td>
<td>1000</td>
<td>340.198</td>
<td>115.734</td>
<td>39.373</td>
<td>13.394</td>
<td>4.557</td>
<td>1.550</td>
<td>0.527</td>
<td>0.179</td>
<td>0.061</td>
</tr>
<tr>
<td>15</td>
<td>1000</td>
<td>314.980</td>
<td>99.213</td>
<td>31.250</td>
<td>9.843</td>
<td>3.100</td>
<td>0.977</td>
<td>0.308</td>
<td>0.097</td>
<td>0.031</td>
</tr>
</tbody>
</table>

The table above shows the values of Hausdorff measure as defined in (15). We approximate the Hausdorff dimension in the same way described in Section 5.4.2. The table reveals this approximation to be between 0.4 and 0.8. To improve this approximation, we can increase the matrix size as well as use a $V$ closer to 0.

6. Appendix

Here we list the MATLAB code referenced throughout the paper.

**Appendix A. Scripts called on in other scripts**

The functions `pointtest.m` and `linetest.m` are called upon in several other scripts.

A.1. `pointtest.m`. This script is called on by `linetest.m`.

```matlab
1 % pointtest.m
2 % Tests to determine whether or not a point under the trace map for
3 % the Thue–Morse sequence if unstable ot not
4 % by WC, RK, MS
5 %----------------------------------------
6 %[fail, test] = pointtest( x0 , v0 , n )
7 %x_0 is out initial x value
8 %v_0 is out initial v value
```
function [ fail, test ] = pointtest( x_0, v_0, n )

fail = -1; \% assumes a value of -1 if it works

\%test is 0 if the point is unstable and 1 else
\%fail is what iteration the point dies

test = 1; \% innocent until proven guilty

\% our function
f = @(x,v)((x^2-2-v);v*(v+4-x^2));

\% relabeling initial conditions
x = x_0;
v = v_0;

\% outside the parabola = instant death
if v<(x^2-4)
    test = 0;
    fail = 0;
end

\% inside the bottom of parabola = instant goodness
if (v<0) && (v>x^2-1)
    test = 1;
    fail = -2;
end

\% in unknown land
\% testing n iterations
if test == 1
    for i = 1:n
        a = f(x,v);
        x = a(1);
        v = a(2);
        \% seeing if the point is unstable
        if (x<-1) && (v+4-x^2>4)
            test = 0;
            \% iteration the point enters the abyss
            fail = i;
            break;
        end
    end
end
end

A.2. linetest.m. This script is called on by boxdim.m, thickness.m, and masterDorff.m.
appendix

B. boxdim.m. This script is called on by masterboxdim.m
%bd1 is the minbox counting dimension!
%bd2 is the maxbox counting dimension!

function [ bd1 bd2 ] = boxdim( V, rk, iter )

% # of iterations for each point
l = iter;

% for the log-log plot
d = [];
N = [];

% Out limits of our cross section
Emin = -sqrt(V^2 + 1) - 1;
Emax = sqrt(V^2 + 1) + 1;

% our actual resolution
n = 2^rk;
% our line at full resolution
cs = linetest(V, Emin, Emax, l, n);
% finding out how many points are in our crosssection is ( n )
u = length(cs);

if sum(cs) == 0
    error('There are no points found in the spectra')
end

% deciding what points to start and end at
a = 2; b = rk - 1;
d = zeros(1,1+b-a);
N = d;

% getting our data for various deltas
% skipping the first point and the last two for funsies
for i = a:b
    % the size of our swath we’re looking at
    res = 2^i;
    % our delta
    d(i+1-a) = (Emax-Emin)/res;
    % starting our j and k counts
    j = 1;
k = 1;
    % as long as we haven’t looked at the whole of ’cs’
while j < u
    % starting at values we haven’t looked at yet
    for k = j:u
        % the first new ’1’
        if cs(k) == 1
            % remember where it is
            answ = k;
            % get out of this silly for loop
            break;
        end
    end
    % update our counts
    j = j + d(i+1-a);
end

% final result
bd1 = N;
bd2 = N + l - 1;
%if there are no more 1's, we're done
if k == u
    break;
end
%our running N_delta count
N(i+1-a) = N(i+1-a) + 1;
%where we start our next search
j = answ + res;
end

%because we care about logs

d = log(d); N = log(N);
%this gives us the slope aka the dimension we want
d = -d;

%partitions our points in half
u = length(d);
fl = floor(u/2);
d1 = d(1:fl); N1 = N(1:fl);
d2 = d(fl+1:u); N2 = N(fl+1:u);
%finds the slope of the best fit line on the plot
p1 = polyfit(d1,N1,1);
p2 = polyfit(d2,N2,1);
%which is our box-counting dimension
bd1 = -p1(1);
bd2 = -p2(1);

%optional code to plot our box counting calculations
%plots log delta vs log N_delta
x1 = d(1):.01:d(fl);
x2 = d(fl+1):.01:d(u);
y1 = p1(1)*x1+p1(2);
y2 = p2(1)*x2+p2(2);

figure; grid;
plot(d,N,'k*',x1,y1,'k',x2,y2,'k')
end

B.2. masterboxdim.m. This script finds the box-counting dimension for various coupling constants.
PROPERTIES OF THE SCHRODINGER OPERATOR WITH THUE-MORSE POTENTIAL

% various coupling constants, \( V \), as \( V \) tends towards zero
% by WC, RK, MS

% This code will call on box dim (which calls on line test (which calls on
% point test))

% \( V \) min and \( V \) max are our \( V \) limits to test
% \( V \) min = 0; \( V \) max = .5;
% points is the number of points we want to test
numpoints = 50;
% All of our \( V \) values to be tested
V = linspace(Vmin, Vmax, numpoints);
% \( V \) = \([10^{-3} 10^{-4} 10^{-5} 10^{-6} 10^{-7} 10^{-8} 10^{-9}]\);
% \( \text{rk} \) is the number of iterations we want to test each dimension calculation
% \( \text{with} \)
% \( \text{rk} = 12; \)
% \( \text{iter} \) is how many iterations we want to test each point for
% \( \text{iter} = 1000; \)
% just a few parameters to make the following program work better :-)
% \( \text{dim} \) presets this as a vector we can build off of
% \( \text{dt} \) allows our plot to update each point as it is found
% \( \text{dim} = []; \)
% \( \text{dt} = 0.01; \)
% declare our figure
figure; grid;
xlabel('\( V \)'); ylabel('\( \text{dim} \)');
hold on
% find dimension for each point, \( V \)
for i = 1:numpoints
% our beautifully calculated dimension(s)
[\( \text{dimmin} \) \( \text{dimmax} \)] = boxdim(V(i), rk, iter);
% remembering our dimension
dim = [dim; [\( \text{dimmin} \) \( \text{dimmax} \)]];
% plot that beast
plot(V(i), \( \text{dimmin} \), 'r*', V(i), \( \text{dimmax} \), 'b*')
pause(dt)
end

APPENDIX C. Thickness

These scripts are used to compute thickness.

% gapFinder.m
% Finds 'gaps' in a vector comprised of 0's and 1's, specifically for
% a cross section of our spectra
%% by WC, RK, MS
%%
% [gap] = gapFinder (cs, lengthCS)
% gap is a 3 x n matrix containing info as described below
% cs is the vector we’re analyzing
% lengthCS is the length of cs
% This program will locate "gaps" (strings of consecutive zeros) in a row
% vector comprised of ones and zeros. It will check each entry of the row
% from left to right, and then output a 3-column matrix. The first
% column of this matrix store the index of the first zero of the i-th gap in
% our row vector. The second column will store the index of the first one
% after the i-th gap. The third column will store the length of the gap,
% calculated by subtracting the index of the zero at the beginning of the gap
% from the index of the one immediately following the gap.

function [gap] = gapFinder (cs, lengthCS)

%Here we define our 3 x n matrix in which we will store our gap
% information.
gap = [ ];

%starting at the second spot of the cross section
for i = 2 : lengthCS
    %if where we are now is a zero and the spot to the left is a 1
    if cs(i) == 1 && cs(i-1) == 1
        %remember this spot, it is the beginning of the gap
        b = i;
    end
    %if where are are now is a 1 and the spot to the left is a 0
    if cs(i) == 1 && cs(i-1) == 0
        %this is the spot to the right of the end of the gap; this allows
        % us to subtract the beginning spot of the gap from this spot and
        % get the actual length of the gap.
        e=i;
        % make a vector with the beginning spot of the gap, the end spot,
        % and the length of the gap
        a=[b e e-b];
        gap = [gap; a];
    end
end

end

%%
% gapEval.m
% Finds the 'bridge' for each 'gap' in a cross section of the spectra
% for each u (the endpoint of a gap) and stores the (length of
% the bridge)/(length of the gap). The minimum of these values
% is the thickness. Calls on gapFinder.m.
% by WC, RK, MS
%
% [ thicku , minThickness ] = gapEval ( cs )
%thick is a 3 x n matrix: the first column is the length of the bridge,
% the second column is the length of the gap, the third is the thickness
% at each u.
%minThickness: the minimum of the third column of thicku, the actual
% thickness
%cs is the vector we’re analyzing
%This program calculates the thickness for each u and takes the minimum
% to calculate the thickness of the cs. The u’s correspond to the
% endpoints of each gap. We work from left to right in the cross section
% and look at each u to find an appropriate 'bridge.' A bridge for u
% contains no point of a gap the same size or larger than the gap that u
% corresponds to. By that definition, if u is the left endpoint, the
% bridge is found by going to the left, and if u is the right endpoint,
% the bridge is found by going to the right.

function [ thicku, minThickness ] = gapEval ( cs )

lengthCS = length(cs);

%to give us a matrix with the endpoints of the gaps & their appropriate
% lengths
[gap] = gapFinder(cs, lengthCS);

thicku = [];
numRows = size(gap);

%to go through all the rows of 'gap'
for k = 1:numRows(1)
    %for the left of the 1st gap
    if k == 1
        %set the bridge = the beginning of the cross section to where
        % the gap begins
        %example if the first gap starts at the 4th spot, all the
        %previous points will be '1's' so they will all account for the
        % bridge thus the length of the bridge will be where we are
        % now (a zero) − 1, in this case the bridge is length 3.
        bridge = gap(k,1)−1;
        %the bridge, the gap length, the thickness for this u
        a = [bridge gap(k,3) bridge/gap(k,3)];
        thicku = [thicku; a];
    end

    %for the right of the last gap
    if k == numRows(1)
        %set the bridge = the last spot of the cs – the last 0 in the
        % gap
        % (gap(k,2) holds the spot to right of the last 0 of the gap
        % so we subtract 1)
        bridge = lengthCS − (gap(k,2)−1);
        %the bridge, the gap length, the thickness for this u
        a = [bridge gap(k,3) bridge/gap(k,3)];
        thicku = [thicku; a];
    end

minThickness = min(thicku(:,3), thicku(:,1));
end

% finding the gaps to the left to make sure we don't go left for the
% 1st gap
if k ~= 1
    % so we look at all the previous rows in the gap matrix
    for l = 1:(k-1)
        % if the length of the last gap is greater than or equal to
        % the gap we are looking at now
        if gap(k-1,3) >= gap(k,3)
            % the bridge = the left end point of the gap we are now
            % - the right endpoint of the gap >= the size of the
            % gap we are now
            bridge = gap(k,1) - gap(k-1,2);
            % the bridge, the gap length, the thickness for this u
            a = [bridge gap(k,3) bridge/gap(k,3)];
            thicku = [thicku; a];
            break;
        end
    end
else if l == (numRows(1) - k)
    % set the bridge = the beginning of the cross
    % section to where the gap begins
    bridge = gap(k,1) - 1;
    % the bridge, the gap length, the thickness for
    % this u
    a = [bridge gap(k,3) bridge/gap(k,3)];
    thicku = [thicku; a];
    break;
end

end

% finding gaps to the right to make sure we don't go right for the
% last gap
if k ~= numRows(1)
    % to look at all the next rows
    for l = 1:(numRows(1) - k)
        % if the length of the next gap to the right is greater than
        % or equal to the gap we are now
        if gap(k+1,3) >= gap(k,3)
            % set the bridge = the left of the gap >= where we are
            % now - the right end point of where we are now
            bridge = gap(k+1,1) - gap(k,2);
            % the bridge, the gap length, the thickness for this u
            a = [bridge gap(k,3) bridge/gap(k,3)];
            thicku = [thicku; a];
            break;
        end
    end
else if l == (numRows(1) - k)
    % set the bridge = the last spot of the cs - the
last 0 of the gap we are now.
\%(gap(k,2) holds the spot of to right of the last
\% 0 of the gap so we subtract 1)
bridge = lengthCS - (gap(k,2)-1);
\%the bridge, the gap length, the thickness for this
\% u
a = [bridge gap(k,3) bridge/gap(k,3)];
\thicku = [thicku; a];
break;
end
end
end

% calculates the actual thickness of the cross section
\minThick = min(thicku(:,3));
end

%------------------------------------
% thickness.m
% Determines the thickness of a given spectra based on a coupling
% constant, V, wanted resolution, rk, and number of tested
% iterations, iter
% by WC, RK, MS
%------------------------------------
%[ thick ] = thickness ( V, rk, iter )
%thick is the value of the thickness
%rk is the RakelSchmugel
%iter is the number of iteration we are testing each point at

function [ thick ] = thickness ( V, rk, iter )
% # of interations for each point
l = iter;
%for the log-log plot
d = [];
N = [];
%Out limits of our cross section
\Emin = -sqrt(V^2 + 1) - 1;
\Emax = sqrt(V^2 + 1) + 1;
% our actual resolution
n = 2^rk;
% our line at full resolution
\cs = linetest(V,Emin,Emax,l,n);
% how long our thing is ( n )
u = length(cs);
%if there are not any good points found at this V
if sum(cs) == 0
    error('There are no points found in the spectra')
end
%if there are all 1's across the V, the thickness is infinity so we plot 0
if sum(cs) == u
    thick = 0;
%if there is a thickness, we want to evaluate it
else
    [thicku, thick] = gapEval(cs);
end
end

%%%%%%%%%%%%%%%%%%%%

% masterthickness.m
% Script that computes the thickness at varying coupling constant
% values, V, and plots the results
% by WC, RK, MS
%%%%%%%%%%%%%%%%%%%%

%Vmin and Vmax are our V limits to test
Vmin = 0; Vmax = .5;

%points is the number of points we want to test
points = 25;

%All of our V values to be tested
V = linspace(Vmin,Vmax,points);

%rk is the number of iterations we want to test each dimension calculation
%with
rk = 17;

%iter is how many iterations we want to test each point for
iter = 15000;

%just stuff to make the following program work better :)
thick = [];
dt = 0.01;

%declare our figure
figure; grid;
xlabel('V'); ylabel('thick');
hold on
%find thickness for each point, V
for i = 1:points
    %our beautifully calculated dimension
    thick(i) = thickness(V(i), rk, iter);
%plot that beast
Appendix D. Hausdorff (Matrix Truncation)

These scripts are used to compute the Hausdorff measure via matrix truncation.

```matlab
% potentialMatrix.m
% This program truncates our operator and determines its eigenvalues
% by WC, RK, MS

%[ S ] = potentialMatrix( n, V )
%S is a vector containing the eigenvalues
%n is the size of our truncated matrix
%V is our coupling constant

function [ S ] = potentialMatrix( n, V )

% This function takes in n, the size of an n x n matrix, and V, a coupling
% constant, and returns the eigenvalues of the n x n laplacian matrix for
% the Schroedinger operator with Thue-Morse potential.

% P will be a row vector containing the Thue-Morse sequence.
% setting our initial point to 1
P(1) = 1;
% 'a' is the number of iterations we want to do on generating our sequence,
% so we will fix 'a' to be 13 so the sequence will be 2^13 = 8192 long
a = 13;
% going through each iteration
for i = 1:a
    % remembering our long our current vector is
    l = length(P);
    % going through each point in our current vector
    for j = 1:l
        % performing bitwise negations
        if P(j)==1
            u(j) = -1;
        else
            u(j) = 1;
        end
    end
    % adding the new part of the sequence to the existing sequence
    P = [P,u];
end
P = P';
% This section populates an n x n matrix with zeros.
D = zeros( n, n );
% Here we populate the sub- and super-diagonals with ones.
```
for j = 1:n-1
    D(j,j+1) = (1);
end

for k = 2:n
    D(k,k-1) = (1);
    % Here we populate the first half of the diagonal with the mirror
    % image of the second half of the diagonal.
    for l = (n/2+1):n
        D(1,1) = V*P(1);
    end
    % Here we populate the second half of the diagonal with our coupling
    % constant V times the i-th entry in the Thue-Morse sequence.
    for l = 1:n/2
        D(1,1) = V*P((n-l+1));
    end
end

end

end

% Here we output the eigenvalues of the Laplacian matrix.
S = eig(D);

%-----------------------------------------------------------------------------
% eigenDorff.m
% Determines the Hausdorff measure at a fixed V value at varying s
% and delta values
% by WC, RK, MS
%-----------------------------------------------------------------------------
%[ table ] = eigenDorff( n, V )
%table is our table of our measure based off of varying s and delta values
%n is the size of our truncated matrix
%V is our coupling constant

function [ table ] = eigenDorff( n, V )

    %n = size of the matrix
    %V = coupling constant

    % vector of eigenvalues
    S = potentialMatrix(n,V);

    % we are letting our interval = [-3;3]

    number=zeros(1,15); % number(i) = number of 1/2^i covers
    covers = 0;

    % checking for 15 different resolutions
    for i=1:15
        res=2^(-1*i); % this is our delta
        % 6*2^-i is our number of maximal covers so we declare a vector with this
```matlab
% many entries to keep track of whether or not we want to include that
% specific cover
% covers = zeros(1, 6 * 2^i);

l = 1;  % this is resetting a value used later on

% looking at each cover and determining whether or not we like it
for j = 2:6 * 2^i
    counter = 0;  % this is zero or one depending on whether a cover is used
    for k = 1:length(S)
        if S(k) >= -3 + (j - 1) * res && S(k) <= -3 + j * res
            % if this point in in our spectrum, we add this cover
            counter = 1;
            % we want to keep track of where this occurs so we do not
            % have to keep looking through this giant vector
            l = k;
            % get out of this silly little for loop
            break;
        end
    end
    % the jth entry correspond to the jth division being used
    covers = covers + counter;
end

% this is the number of covers we are using
number(i) = (covers);
covers = 0;
end

% all of our 's' values we want to test
s = linspace(0, 1, 10);
% creating a matrix with 10 rows and the number of columns corresponding to
% the number of s's we are testing
table = zeros(10, length(s));
% computing the rows of our matrix
for i = 1:15
    for j = 1:length(s)
        % using the formula where the measure = n * delta ^ s
        table(i, j) = number(i) * (2^(-1*i))^s(j);
    end
end
end
```
These scripts are used to compute the Hausdorff measure via trace map iterations.

```matlab
function [H] = deltaDorff (Emin, Emax, cs, perDelta, s)
    %cs = linetest (V, Emin, Emax, iter, rk); %calls on linetest to get a cross %section at a specific V
    %the length of the cross section
    lengthCS = length(cs);
    %because our spectra exists at the endpoints of the parabola y = x^2−4, we
    %want the endpoints of the cross section to be 1 for "in the spectra."
    %The line test function tests the values on the parabola and rules them out %due to numerical error.
    cs(1)=1; cs(lengthCS)=1;
    %this is defining a ~x3 matrix. It will store the places and lengths %of delta) of strings of 1's in the cross section.
    deltaU = [ ];
    %this accounts for if cs(1) = 1 && cs(2) = 0
    b = 1;
    for i = 2 : lengthCS
        %if we are at a 1, and the last spot was a 0
        if cs(i) == 1 & cs(i-1) == 0
            %remember where we are now. it is the beginning of the covering
            b = i;
        end
        %if we got to a zero, but the last space was a 1
        if cs(i) == 0 & cs(i-1) == 1
            %remember where are are right now
            e=i;
            %store where we began, ended, and the length
```
a=[b e e−b];
%add the new information to the running matrix
deltaU = [deltaU; a];
end

%this accounts for the last covering
if i == lengthCS
%remember the place to the right of the last covering
e=i+1;
%store where we began, ended, and the length
a=[b e e−b];
%add the new information to the running matrix
deltaU = [deltaU; a];
end
end

h = [];
SIZE = size(deltaU);

%to go through all the rows of deltaU
for i = 1:SIZE(1)
%if the delta covering is less than what we want, leave it along
if deltaU(i,3) <= (lengthCS*perDelta)
    h = [h; deltaU(i,:)];
%if the delta covering is more than what we want, split it up
else
    %how many divisions we can make
    c = floor((deltaU(i,3))/(lengthCS*perDelta));
    %storing the divisions
    for j = 0:c−1
        start = deltaU(i,1)+(lengthCS*perDelta)*j;
        finish = start + lengthCS * perDelta;
        a = [start finish finish−start];
        h = [h; a];
    end
    %storing the left overs
    if mod(c,lengthCS*perDelta) ~= 0;
        start = finish;
        finish = deltaU(i,2);
        a = [start finish finish−start];
        h = [h; a];
    end
end
end
%calculate the hausdorff measure
H = sum((h(:,3)*perDelta*Emax−Emin).^s);
% masterDorff.m
% For a fixed coupling constant, V, this finds the Hausdorff measure
% for the spectra with varying 's' values
% by WC, RK, MS

% The coupling constant
V = 0.03;
% number of iteration for each point
iter = 10000;
% the resolution (2^rk)
rk = 15;
% our limits
Emin = -sqrt(V^2+1)-1;
Emax = sqrt(V^2+1)+1;
% the 's' values we're testing
s = linspace(0,1,10);
% makes things nice
dt = 0.001;
d = [];

% declaring our figure and its properties
figure; hold on; grid;
set(gca,'FontSize',12);
title(['\{\delta\} vs. Hausdorff Measure at a V = ', num2str(V), '']);
xlabel('\{\delta\}'); ylabel('Hausdorff Measure');
axis([0 .001 0 10]);

% the vector we're testing
cs = linetest(V,Emin,Emax,iter,2^rk);

A = zeros(rk,length(s));

for j = 1:length(s)
    for i = 1:rk
        % our delta
        perDelta = 2^(-(i+10));
        d(i) = (Emax - Emin) * perDelta;
        % our Hausdorff measure with specified s and delta values
        H = deltaDorff(Emin,Emax,cs,perDelta,s(j));
        haus(i) = H;
        A(i,j) = H;
    end
    % plot(d, haus);
    % pause(dt);
end
%
% plotting
% plot(d,A(:,1),'k',d,A(:,2),'k',d,A(:,3),'k-',d,A(:,4),'k',d,A(:,5),'k:',d,A(:,6)
% 'k',A(:,7),'k--',A(:,8),'k',A(:,9),'k:',A(:,10),'k')
%
% putting the legend
% legend ('s = ', num2str(s(1)), ', ' s = ', num2str(s(2)), ', ' s = ', num2str(s(3)), ', ' s = ', num2str(s(4)), ', ' s = ', num2str(s(5)), ', ' s = ', num2str(s(6)), ', ' s = ', num2str(s(7)), ', ' s = ', num2str(s(8)), ', ' s = ', num2str(s(9)), ', ' s = ', num2str(s(10)))
References


Department of Mechanical Engineering, Ohio University, Athens, OH 45701
E-mail address: wc565210@ohio.edu

Department of Mathematics, St. John Fisher College, Rochester, NY 14618
E-mail address: rmk09830@sjfc.edu

Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803
E-mail address: mston16@lsu.edu