5-2015

Loewner Space-filling Curves

Hannah Marie Clark
hclark9@vols.utk.edu

Loewner Space-filling Curves

Hannah Clark

May 4, 2015

Introduction

Is there such a thing as a curve you cannot draw? Perhaps not, but space-filling curves are the next best thing. Space-filling curves are continuous curves that touch every point in a region. If that region is a square, then the curve looks like a filled-in square. However, not much can be discovered from a region that is completely filled. Instead, we are looking at sequences of regular curves that are space-filling in the limit. In particular, we are exploring how we can use the Loewner Equation to encode our 2-dimensional curves into continuous, 1-dimensional functions and understand how geometric data of the curves is converted to analytic information of the functions.

The outline of this paper is as follows: In the first section, we will set the stage by introducing the Loewner equation and the context for the problem at the heart of our research. Our work focuses on space-filling curves, a concept we will introduce in Section 2 along with the key space-filling curve examples we will analyze. Section 3 contains the results of using the Loewner equation together with our space-filling curve examples.

Acknowledgements: We would like to thank Dr. Joan Lind for her advising, and NSF for its support.

Contents

1 Loewner Equation **2**

2 Space-filling Curves **7**

3 Results **11**

4 Bibliography **17**

1 Loewner Equation

The Loewner differential equation was developed in 1923 by Charles Loewner to solve problems in complex analysis. But it has come back en vogue due to a recently-uncovered connection with probability. In 2000, Oded Schramm discovered that if Brownian motion is used as the Loewner Equation's driving function, $\lambda(t)$, then the equation generates an interesting family of random curves called Schramm-Loewner Evolution, or SLE. **Example 4** demonstrates using Brownian motion as the driving function. SLE has been used to prove a number of prominent conjectures in probability, and it has rekindled interest in this equation. This 100-year-old equation is more versatile than originally thought, which is our motivation for looking into it more.

Let's start by discussing the following version of this equation, called the **chordal Loewner equation**:

$$\frac{\partial}{\partial t} g_t(z) = \frac{2}{g_t(z) - \lambda(t)}, \;\; g_0(z) = z \tag{1}$$

Here, $\lambda(t)$ is a continuous, real-valued function and z is a complex number. The behavior of this simple-looking initial value problem depends on the choice of the function $\lambda(t)$. So, in a sense, $\lambda(t)$ *drives* the behavior of the whole equation. For this reason, we call λ the **driving function**. The other feature to note is that we also get to pick a complex number z to be our initial condition and analyze its flow under the differential equation.

Plugging in a driving function λ, the output of the Loewner Equation (LE) is a continuous curve. But the curve does not just magically appear. According to [3], a unique solution $g_t(z)$ exists on some time interval, by the existence and uniqueness theorem for differential equations. The solution $g_t(z)$ always exists except where problem spots occur. In (1), the only problem spots occur when the denominator is zero. As we flow through the equation, we collect all the points that make the denominator zero and add them to our output curve. When all of these solutions at the problem spots are taken together, we define the hull at time t, denoted K_t, as the collection of initial points that lead to solutions at problem spots. In other words,

$$K_t = \{z_0 \in \overline{\mathbb{H}} : z(s) = \lambda(s) \text{ for some } s \leq t\}.$$

This family of hulls, $\{K_t\}_{t \in [0,T]}$, is the increasing family of sets which is the output curve for $\lambda(t)$ via the Loewner equation. λ is the driving function, and K_t is generated by λ.

On the other hand, if we know what curve the LE outputs (or K_t) we can work backwards and find the driving function that creates that curve. We refer to this process as "encoding". The LE has one-to-one correspondence and is capable of encoding curves into functions and functions into curves. All of the examples included in this paper have the following form:

$$\text{continuous curve} \overset{LE}{\longleftrightarrow} \text{driving function}$$

Although the LE looks fairly simple, it cannot be solved explicitly for most driving functions that we choose. Some driving functions for which it can be solved explicitly are constant functions, linear functions, and square root functions [1]. Whether we can explicitly solve the equation or not, we use a Java program called BLEAT to help us at least approximate what the output curves are for different driving functions. BLEAT was developed by a group of students at Belmont University, and it stands for The Belmont Loewner Equation Analysis Tool.

BLEAT has the LE built in to act as a "black box" into which we can input continuous, real-valued functions $\lambda(t)$, and it outputs the hull or growing curve in the complex plane. Because the LE is one-to-one, we can also input a curve and it will output a graphical representation of the driving function λ. Many of our visual examples of driving functions and curves are generated using BLEAT, so it should become more clear how the "black box" operates. All the features of this Java program can be learned about by reading [1].

Example 1. As an example of a scenario where we know the driving function λ but not the curve that gets generated, we will take $\lambda(t) = \sin(10t)$ as the driving function. In Figure 1, $\sin(10t)$ is graphed on the right. But what curve will the LE via BLEAT give us with $\sin(10t)$ as the driving function? That is the image on the left. Plugging $\sin(10t)$ into the LE for λ outputs the curve on the left. If we knew the equation for the curve on the left, we could plug that into the LE via BLEAT and recover $\sin(10t)$ as the output, but we do not actually know the equation for the curve in this case.

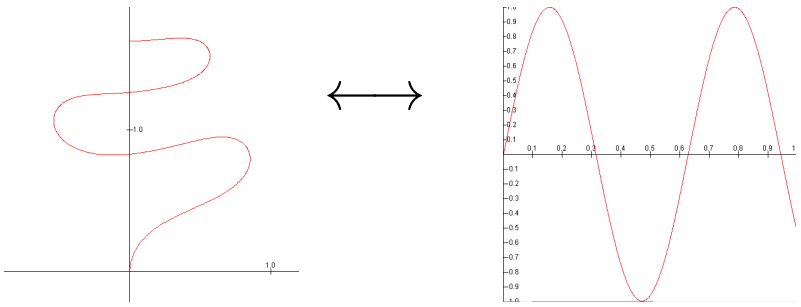


Figure 1: $\lambda(t) = \sin(10t)$ is shown on the right. The corresponding Loewner curve is shown on the left.

But what is so special about the curve that $\sin(10t)$ gave us? That is something we will try to discover by looking at many examples and trying to figure out more about the correspondence between the two images.

Example 2. Here we are using $8\sqrt{t}$ as the driving function, which is graphed on the right of Figure 2. Functions of the form $c\sqrt{t}$ are helpful because we can plug that into the LE and calculate with relative ease what the equation of the output curve is. Inputting functions of this form results in outputs that are line segments, such as the line segment you see on the left of Figure 2. The angle of the line segment depends on the value we choose for c. If $c = 0$, then the segment will be a vertical line. Since in our example, $c = 8$, the line segment is intersecting at an angle much less than 90 degrees.

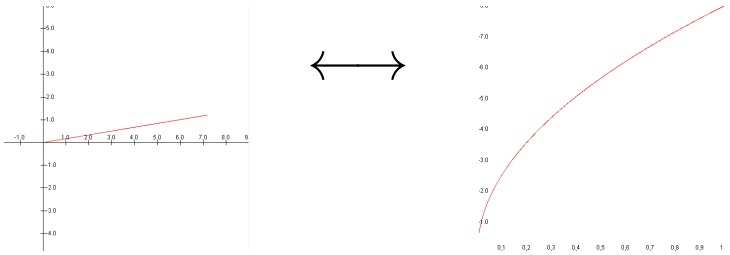


Figure 2: $\lambda(t) = 8\sqrt{t}$ is shown on the right. The corresponding Loewner curve is shown on the left.

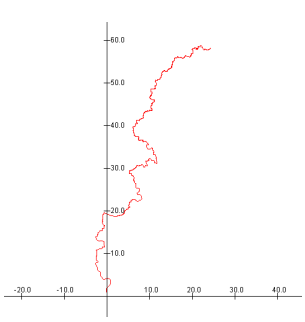
Example 3. Another interesting family of driving functions is the family $\lambda(t) = c\sqrt{1-t}$. This family has different behavior for $c < 4$ and $c \geq 4$. When $c < 4$, the curve that is generated spirals very tightly to its final point. But the spiral is so tight that it is hard to see visually. Figure 3a shows the curve generated by the driving function $\lambda(t) = 2\sqrt{1-t}$. On the other hand, when $c \geq 4$, the curve that is generated hits back on the real line, and creates a "bubble" effect. This can be seen in Figure 3b, where the curve is generated when $\lambda(t) = 8\sqrt{1-t}$. This doubling back phenomenon is discussed more in the next example. [1]

(a) $\lambda(t) = 2\sqrt{1-t} \longrightarrow$ a tight spiral. (b) $\lambda(t) = 8\sqrt{1-t}$ doubles back.

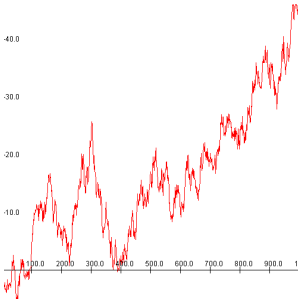
Figure 3: The different results generated using $\lambda(t) = c\sqrt{1-t}$.

Example 4. Earlier we referred to the fact that using a multiple of Brownian motion as the driving function in the LE has been very important in probability. A whole family of random curves can be generated that are referred to as SLE. In [3], we learn that SLE_κ is the random curve generated by the LE. More specifically, the curve is generated by $\lambda(t) = \sqrt{\kappa}B_t$, where B_t is Brownian motion.

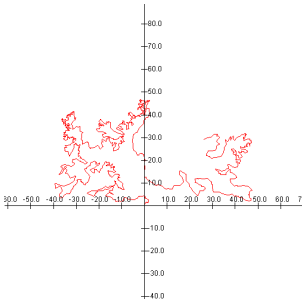
This family has phase transitions, as shown in [7]. When $\kappa \leq 4$, the hulls are simple curves. This type of curve can be seen in Figure 4a. Simple curves are not generated for $\kappa > 4$. Specifically, when $4 < \kappa < 8$ the hull is a random curve that forms bubbles. In other words, the curve hits back on either itself or the real line. This phenomenon can be seen in Figure 4c. Lastly, when $\kappa \geq 8$ there is another phase transition, and the hulls become space-filling curves. We will discuss space-filling curves in Section 2.



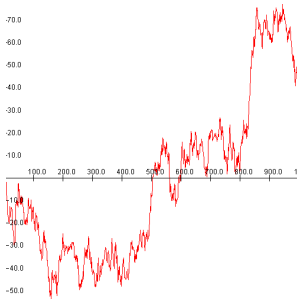
(a) SLE_1



(b) a sample of Brownian motion B_t



(c) SLE_6



(d) a sample of $\sqrt{6}B_t$

Figure 4

Obviously from Figure 4b and Figure 4d, the driving functions that produce the curves are fairly wild. Oftentimes we will not be able to work out the equation of the curve given the driving function. Consequently, we often rely on BLEAT to give us graphical approximations of the data. So, when we have only approximations of complicated curves, how can we learn more about how the two images relate to each other?

One important tool we use to analyze how the images of curve and function relate is the **Lip(1/2) Norm**. It is a measurement of the driving function that gives information about the geometry of a curve. A more formal definition follows:

Definition. λ is in Lip(1/2) if there is a $M > 0$ such that for all $t \neq s$ in the domain of λ,

$$|\lambda(t) - \lambda(s)| \leq M\sqrt{|t-s|}. \tag{2}$$

The smallest value of M is called the **Lip(1/2) Norm** of λ.

For our purposes, it is easier to work with the following equation, which is equivalent to Equation (2).

$$M = \sup\left\{\frac{|\lambda(t) - \lambda(s)|}{\sqrt{|t-s|}}, \text{ for t} \neq \text{s}\right\}. \tag{3}$$

This measurement is helpful to us in our efforts to discover the relationship between the driving function and curve. We will start with it's utility on the drving function side.

Lip(1/2) functions form a natural class of driving functions for the LE. The Lip(1/2) norm (the M Value) that satisfies Equation (3) is a good measure of what is happening in the driving function because it is taking data points from the driving function and comparing them to each other. In Section 3 we discuss a MATLAB code we made which pulls different points t and s from a finite list of data points from the driving function and compares them against one another until Equation (3) is satisfied.

But we are also interested in how the information from the Lip(1/2) norm translates to the curve. The Lip(1/2) norm tells us if the curve could be space-filling or not. In the SLE_κ case (Figure 4), the constant κ from $\lambda(t) = \sqrt{\kappa}B_t$ tells us about the geometry of SLE_κ. In SLE_κ, the phase transitions occur at $\kappa = 4$ and $\kappa = 8$, when the geometric behavior changes. The behavior of the curves in Figures 4a and 4c is very similar to the behavior of curves driven by functions $\lambda \in$ Lip(1/2). The Lip(1/2) norm acts as κ.

To expand on this, we have the following theorems:

Theorem 1. If the Lip(1/2) norm is less than 4, we have a simple curve. (Lind, Marshall, Rohde)

Think back to Figure 3a. We mentioned that to generate that curve we used $c = 2$ in the driving function $\lambda(t) = c\sqrt{1-t}$. For that family of driving functions, the Lip(1/2) norm is c. So, the Lip(1/2) norm for Figure 3a is 2. Since 2 is less than 4, Figure 3a is an example of a simple curve.

Theorem 2. If a curve is space-filling, then the Lip(1/2) norm is greater than or equal to 4.0001. [4]

However, 4.0001 is not the correct number, in the sense that it is not where the phase transition occurs. It is only a lower bound. In other words, at each time t that λ hits the real line, the local Lip(1/2) norm is at least 4. But 4.0001 is just the lower bound for the curve to be space-filling.

Our Problem: What is the biggest value of M so that the following is true? *If a curve is space-filling, then the Lip(1/2) norm of the driving function is at least M.*

We will formulate an approach to this problem by analyzing examples. We want to find what the correct number is by finding Lip(1/2) norms of space-filling curves and coming down from above to try to get as close as we can to 4.0001.

2 Space-filling Curves

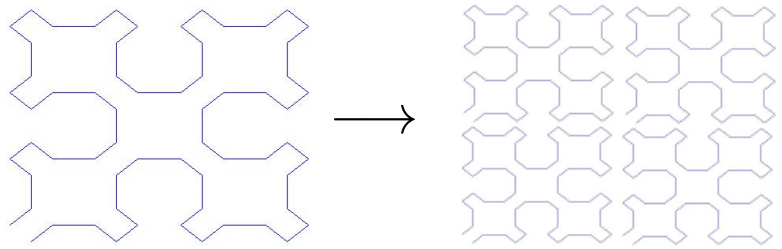
Space-filling curves are continuous curves that touch every point in a region. If we were to include a picture of a space-filling curve, it would just look like a two-dimensional blob. Every point in the region would be touched, and it would appear to be completely filled in. Instead, we are including images of sequences of regular curves that are space-filling in their limit. A more formal definition follows:

Definition. A space-filling curve is a continuous function $\mathbf{f}$ from [0, 1] onto $\mathbf{A}$, where $\mathbf{A}$ is a region in $\mathbb{R}^2$ with positive area.

In this paper, we will focus on two fascinating space-filling curves. For more information on the construction of space-filling curves in general and for practice with the specific example of how to construct the Peano space-filling curve, refer to *Topology: A First Course* by James R. Munkres, seen as [6] in the Bibliography.

Sierpiński Example. Our first example is an historic example of a space-filling curve called the Sierpiński curve. We can create this space-filling curve by taking the limit of "nice" or non-space-filling curves defined in the unit square.

We start with the image in Figure 5a. That is the building block we will use to get to the image in Figure 5b. If we shrink Figure 5a down, copy it three times, and attach those three copies to the original "building block" then we get 5b. If you look closely you can tell that 5b is not continuous. A few of the connections are broken between the copies. For the space to truly get filled up, we should reconnect the spots that have been broken in such a way that the four pieces fit together in a continuous manner. The continuous version can be seen in Level Two of Figure 6.

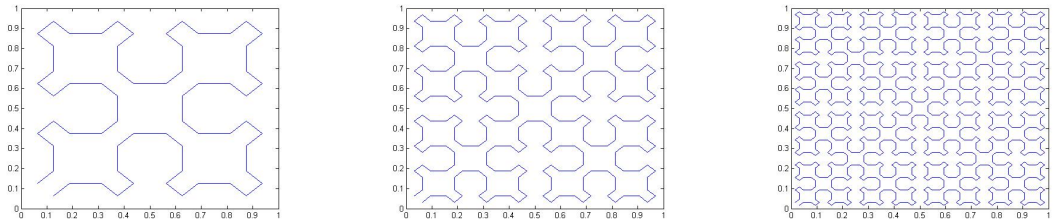


(a) "building block" of Sierpiński curve

(b) discontinuous version of Level Two Sierpiński curve

Figure 5

Now we use Level Two of Figure 6 as our new building block. We shrink that image down and attach three copies of that image to itself, again being careful to reconnect the center points to fit together in a continuous manner. We should then end up with the Level Three image. If this process is continued an infinite number of times, then we have created a space-filling curve in the limit. Some early levels of the curve are seen in Figure 6, generated with a MATLAB code we developed that can compute this specific curve at any complexity.



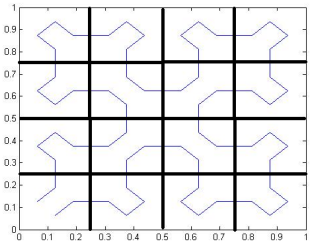
(a) Level One

(b) Level Two

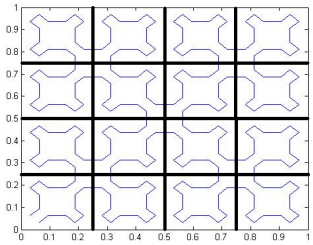
(c) Level Three

Figure 6: Sierpiński curve created in MATLAB

To build the MATLAB code for this curve, the first step is to break Level One and Level Two down into a series of smaller square pictures and determine if each picture is replaced by a more detailed version of the same picture in the next level, without any abnormalities.

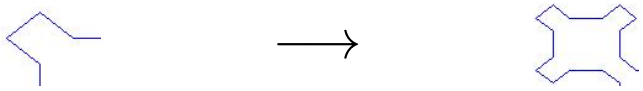


Level One

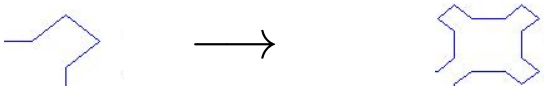


Level Two

Below is a close-up view of the upper-left pictures in Level One and Two. Every square picture in Level One that looks like the image on the left gets replaced by the image on the right every time.

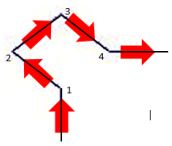


Another example is the upper-right picture in Levels One and Two. Every square in Level One that looks like the image on the left, below, gets replaced by the image on the right, below. There are no exceptions to this.



Since there are not any abnormalities or exceptions, we are able to track the pictures in Level Two that replace each picture from Level One. Now that we know this replacement scheme works without any unusual features, we are able to generalize this result for any level. This is the first step to ensuring we can program this curve into MATLAB.

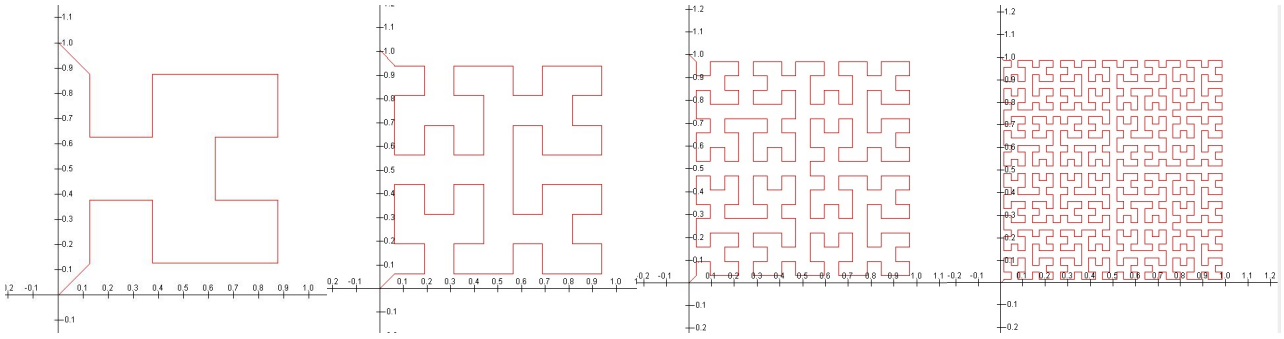
Once it is determined that the replacement scheme works, we encode each square picture into a sequence of turning points. This is important because it enables us to tell the code in what order the points should be hit and even in what order to hit each square picture.



Now that we know the replacement scheme and what order to move through the squares, we have the information we need to be able to tell MATLAB how to generate this specific curve, the Sierpiński curve, at any level. Attached in the Appendix is the code we use to create this curve.

Hilbert Example. Another historic space-filling curve that has features similar to those of the Sierpiński curve is the Hilbert space-filling curve, seen below in Figure 7. It is formed in the same way, taking the initial image, shrinking it down and attaching other versions to it. However, this curve is different from the Sierpiński curve in that sometimes the images need to be rotated in order to fit together continuously.

Level Two in Figure 7 shows our initial "building block" image. Level Three displays the building block with three other versions attached to it in such a way that Level Three is continuous. In order to be continuous, the initial image from Level Two has been rotated from laying on its side to an upright position. So now, in Level Three, the shape closest to the origin looks more like a field goal in football. Levels Four and Five continue this pattern, as do all future levels, because this curve needs rotation in order to be continuous and ultimately space-filling in the limit.



(a) Level Two (b) Level Three (c) Level Four (d) Level Five

Figure 7: Plots of Hilbert Curve

In [4] it is shown that the driving function for the Hilbert curve is Lip(1/2). This means that equation (2) is satisfied for every t and s value in the domain of λ for the Hilbert curve. In fact, the Lip(1/2) equation holds true for every value of the Sierpiński curve as well. However, the work is theoretical and does not provide a way of computing the Lip(1/2) norm. We have developed a MATLAB code to compute the Lip(1/2) norm, which we will talk more about in Section 3.

3 Results

We have now introduced the class of examples that we are going to use to try to get close to the 4.0001 number from **Theorem 2**: space-filling curves. Space-filling curves are good examples of continuous curves that we can control and convert into analytic data via the Loewner Equation.

The picture on the left of Figure 8 is the Level Two Sierpiński curve, and a graphical approximation of the driving function is on the right. Since we have the MATLAB code that generates the Sierpiński Curve, we are able to have total control over the image on the left. What we do not know is the driving function λ that could be plugged into the LE to generate that curve. Since BLEAT has the LE built in, it is able to give us the driving function, but just in graphical form, which is the image on the right. If we came from the opposite angle and instead had total control over the driving function that would generate the curve on the left, we could input that and the curve on the left would be the output.

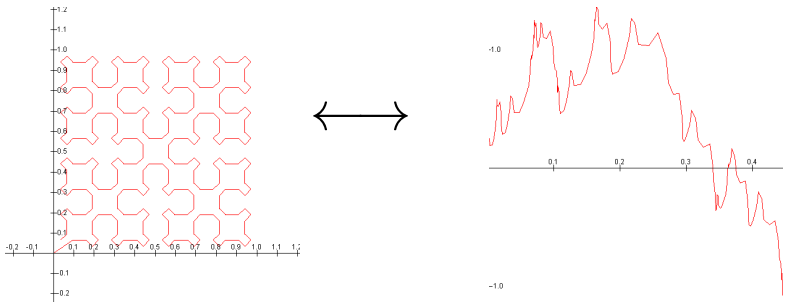


Figure 8: Level Two Sierpiński curve and corresponding driving function

We now have the examples we will focus on of curves that get encoded into functions via the LE. But there is still so much to learn. *How do geometric properties from the curve get converted into analytic data of the driving function, and vice versa?*

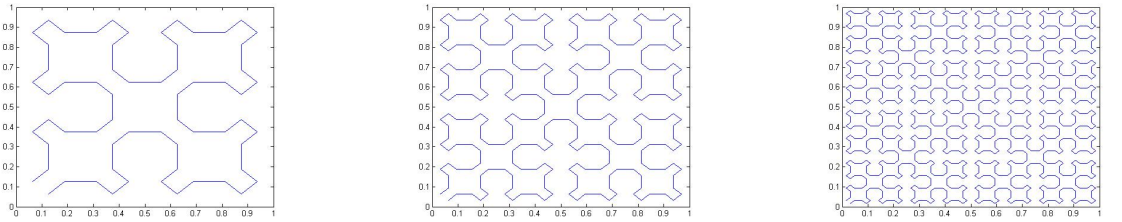
Since the Lip(1/2) norm is a measurement of the "space-fillingness" of the curve while also analyzing different values of the driving function, it is a very valuable metric for understanding the relationship between curve and driving function. Our method to learn more about the encoding process is to start with the space-filling curves we have previously mentioned, the Sierpiński and Hilbert curves, and calculate their Lip(1/2) norms. Since the curves are space-filling, we know that their Lip(1/2) norms will be somewhere above 4.0001. We also know that 4.0001 is not the correct value where space-filling begins (**Theorem 2**), and our examples will hopefully get us close to what the correct value should be as we approach it from above.

But finding solutions to the Lip(1/2) equation, Equation (2), by hand would be inconvenient at best. Therefore we created a MATLAB code to produce an approximation to the Lip(1/2) norm of any curve. We have a finite list of data points from the driving function. Our code pulls from those points two at a time, say points t and s, and compares them to each other in the equation

$$M = \sup \left\{ \frac{|\lambda(t) - \lambda(s)|}{\sqrt{|t-s|}}, \text{ for t} \neq \text{s} \right\}. \tag{3}$$

The code is attached in the Appendix.

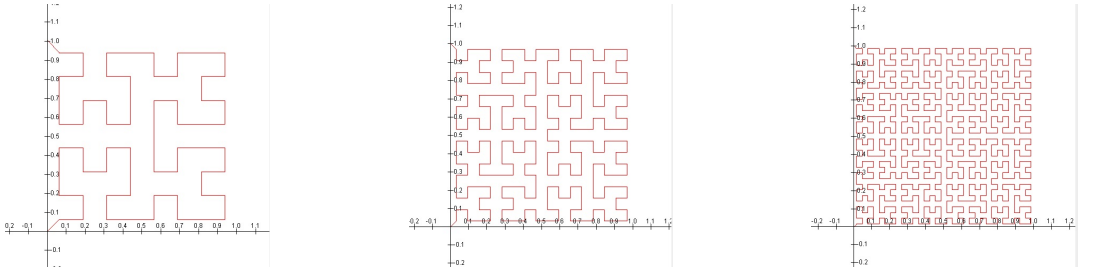
We first made our code give us the M value of many different iterations of the Sierpiński and Hilbert curves. Figure 9 shows the M values for the Sierpiński curve, and Figure 10 shows the values for the Hilbert curve. Our theory is that the limit of the M values, or the Lip(1/2) norms, is an approximation for the Lip(1/2) norm of the actual space-filling curve.



(a) M= 8.222268678981 (b) M = 8.594048361706 (c) M = 8.629972013631

Figure 9: Lip(1/2) norms of the Sierpiński curve

The goal is that through these examples we will gain insights into how space-filling curves are encoded and what the most accurate Lip(1/2) norm is when curves become space-filling. We want M to be as close to 4.0001 as possible.

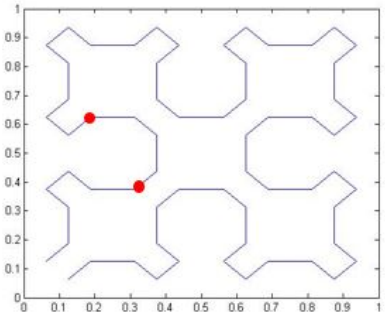


(a) M = 11.886039034799 (b) M = 11.788168604022 (c) M = 11.886039035128

Figure 10: Hilbert Curve Lip(1/2) Norms

The Lip(1/2) norms (or M values) for the Sierpiński curve are around 8, whereas the M values for the Hilbert curve in Figure 10 are greater than 11. So we see that the Sierpiński curve is a better example than the Hilbert curve for our purposes.

Since the Sierpiński curve is our best example, we want to know more about it. For instance, what points t and s in Equation (3) is the code pulling from to generate $M = 8.629972013631$ in Level One? Where do those points correspond on our curve? The dots on the curve in Figure 11 show the location from which the t and s values are pulled to give us our approximation of the Lip(1/2) norm for this level of the curve.



M	t	s
8.22226867898107	55	59
8.04052441901347	55	58
7.56260107504751	55	57
7.21497153873538	55	60
6.64159457455441	63	66
6.55255480102561	55	56
6.24258053163201	54	60
6.19382747190232	63	65
6.16704052252734	51	66
6.14141975213342	13	14

Figure 11: Top 10 M Values for Level One of the Sierpiński Curve

Because those dotted points are so close together, there is a lot of approximation error. The way the curve is generated, it starts at the bottom left corner and builds counter-clockwise. By the time the curve generates to the point of the dots, so much of the curve has already been constructed that this area is very shadowed and more difficult to compute. Plus, the points give information about the geometry of the curve between them, and when the points are so close, it does not give a good representation of the geometry of the total curve. Ideally, we would prefer the points be pulled from farther apart.

Figure 9a only shows us the Lip(1/2) norm, or the supremum of Equation (3). What can be learned from looking at, for instance, the top ten M values and the points from which they are generated on the curve? See Figure 11.

The dots in Figure 11 correspond to the 55th and 59th data points of the curve. In other words, the t and s points that Equation (3) pulls from to generate the first M value are points 55 and 59 (where there is a lot of approximation error). Note that the top ten values all generally want to stay in that same area.

Another trend we noted in many of the curves we analyzed was the tendency for the Lip(1/2) norm to be pulled from points that were exactly four apart from each other, such as 55 and 59. For this reason, we altered our MATLAB code to pull from points that were at least five apart. In doing that, the top five values from the example in Figure 11 changed to the following:

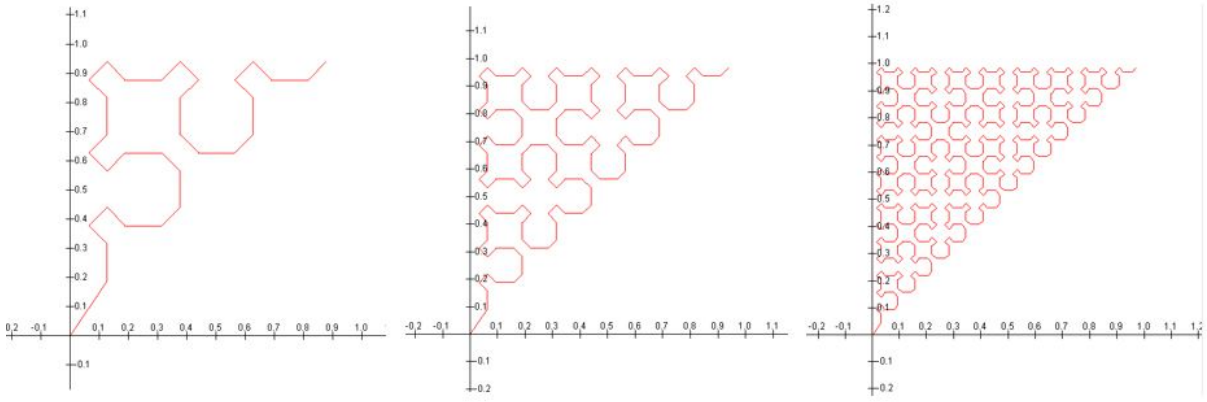
M	t	s
7.21497153873538	55	60
6.24258053163201	54	60
6.16704052252734	51	66
6.07978484444709	54	59
6.0672502539983	12	18

As you can see, some of these values are the same as those generated in Figure 11. It makes sense that the new largest M value would be the first one from the list in Figure 11 that also satisfies the new criteria of being at least five points apart. In fact, the new M value is exactly five points apart.

That was the trend we saw: when we told MATLAB to pull from at least six points apart, the first value was *exactly* six apart. And it always stayed in the dotted range as best it could. Something about the geometry of that area is appealing for the Lip(1/2) norm to stay in.

But although the Lip(1/2) norm is smaller when we force the program to pull from points that are farther apart, we are forcing the Lip(1/2) norm to change. The natural Lip(1/2) norm is still 8.222 from Figure 11, and that is the one we are focusing on, even though the data is not pulled from our ideal location.

After learning all we could about the regular Sierpiński curve, we altered our MATLAB code to form in reverse. In other words, the curve now forms in a clockwise direction. Unfortunately, when we plugged in our data to BLEAT, it would get stuck, seen in Figure 12.



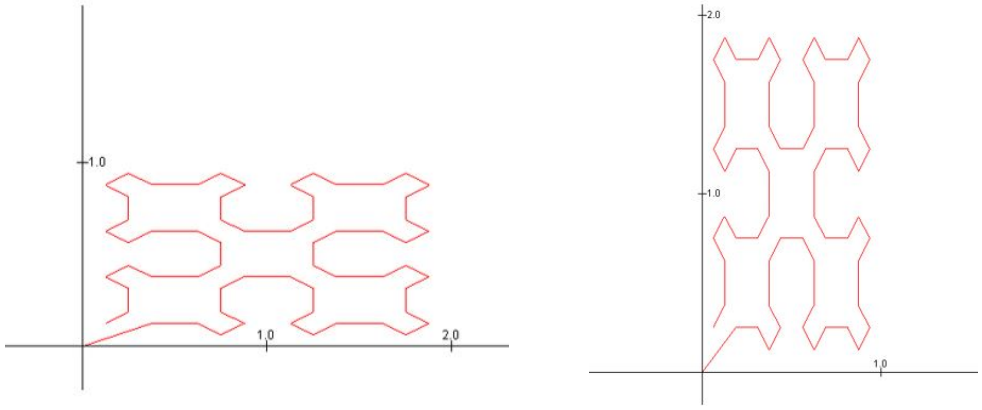
M = 5.95006406061327 M = 7.95639102039623 M = 8.56043925525413

Figure 12: Lip(1/2) Norms of half of the Sierpiński curve generated in reverse

As we mentioned earlier, as the curve is formed in BLEAT, the later points are sometimes forming in a shadow under the curve, which BLEAT has a hard time overcoming. A little over halfway through the generation of the reverse curve, BLEAT gets stuck. This is not going to give us the information we are looking for about the Lip(1/2) norm of the total space-filling curve, and so we moved on to other ideas.

But before moving on we did check what the Lip(1/2) norms are for these levels. Since BLEAT got stuck a little over halfway each time, we stopped the generation at exactly halfway. Athough the Lip(1/2) norms are in fact smaller (closer to 4.0001) in this Figure, they are not representative of the whole Sierpiński curve, and so we do not consider the values in Figure 12 to give us the most accurate data. But perhaps we can manipulate the original curve in other ways to get the Lip(1/2) norm down without BLEAT getting stuck.

Back to the regular Sierpiński curve. Thus far we have dealt with this curve in the unit square. What happens to the Lip(1/2) norm if we alter the region? Our first attempt to change the region is to stretch the different levels of our curve out to 2 in either the x or y direction to see if a conformal transformation of the square will get us closer to the value that 4.0001 is supposed to be. If you recall, the Lip(1/2) norm of the Level One Sierpiński curve in Figure 11 is 8.222. Let's see how that gets altered by stretching.



(a) M = 9.26335798452393 (b) M = 10.3082076129054

Figure 13: Lip(1/2) Norms when x and y are multiplied by 2

Figure 13 shows the Lip(1/2) norms generated by stretching the original unit square region. Figure 13a tells us that stetching out to 2 in the x direction gives us a worse approximation for the Lip(1/2) norm, because our original value of 8.222 gets bumped up to 9.263. This is not the direction we want to go in order to approach 4.0001. Figure 13b is even worse, jumping up to 10.308. We also tried stretching to some other values between 1 and 2. For some more complex iterations, stretching to 2 caused the curve to get stuck. Stretching to 1.5 often worked but still did not help us in our quest to find the correct value for **Theorem 2**.

So far in our efforts to find the the correct value in the Theorem, our best estimate is the regular Sierpiński curve from Figure 9. But we have only just begun exploring the family of space-filling curves that could be studied. Perhaps performing different conformal transformations of the unit square on these historic curves will be beneficial, or maybe the next step should be looking at other curves entirely. With more examples in our arsenal, we hope to be able to develop a theoretical proof as to what the correct value in **Theorem 2** should be.

Note: All of the experiments and manipulations we are showing are of the Sierpiński curve. We tried the different tests on all the curves (Hilbert, Sierpiński, and reverse Sierpiński) but because the regular Sierpiński curve is our best example, that is the data included. However, attached in the Appendix is a file with all of our collected data.

4 Bibliography

[1] S. Claiborne, C. Simpson, BLEAT: The Curve-Creating Black Box, *Belmont Univ. Math.* preprint (2010).

[2] J. Lind, A sharp condition for the Loewner equation to generate slits, *Ann. Acad. Sci. Fenn. Math.* **30**, 143–158 (2005).

[3] J. Lind, J. Robins, Loewner Deformations driven by the Weierstrass function, *Univ. of Tenn. Knox. Math.* preprint (2015).

[4] J. Lind, S. Rohde, Spacefilling curves and phases of the Loewner equation, *Indiana Univ. Math. J.* **61**, 2231–2249 (2012).

[5] D.E. Marshall, S. Rohde, The Loewner differential equation and slit mappings, *J. Amer. Math. Soc.* **18**, 763–778 (2005).

[6] J. Munkres, Topology: A first course, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1975).

[7] S. Rohde, O. Schramm, Basic properies of SLE, *Ann. Math.* **161**, 879-920 (2005).