A Study of Cutting Sequences of Eigenvector Trajectories on the Square Grid under Repeated Linear Maps by a Hyperbolic Transformation Matrix

Athreya Murali

Senior Exhibition

Rip Blackstone: Advisor
John Caballero: Second Faculty Reader
Dr. Martin J. Schmoll, PhD: Outside Expert
Riley Haywood: Sophomore

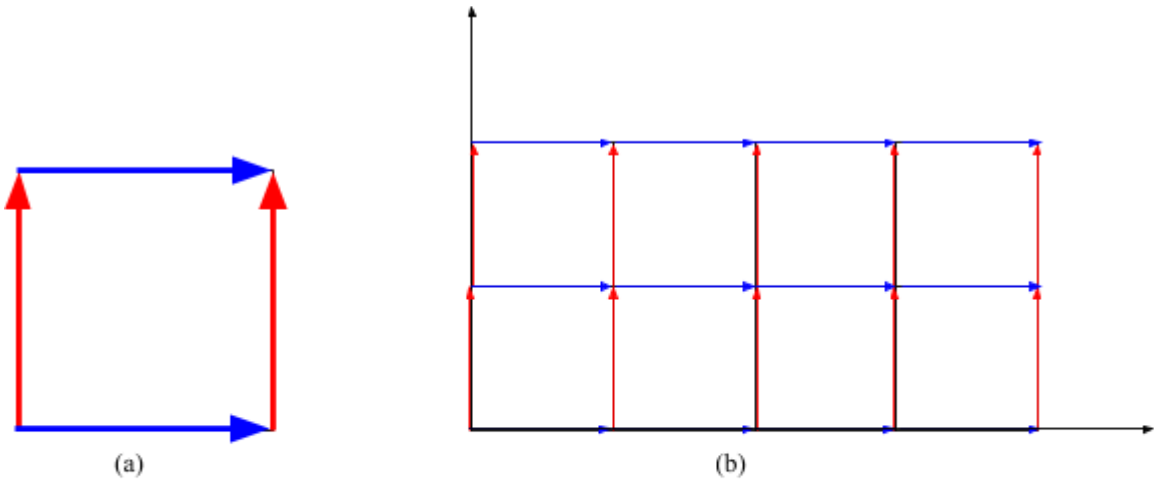
28 Mar 2018

The intricate field of mathematical billiards is based on a surprisingly simple scenario: one is given a simple closed curve M (called the billiard table) and a point-mass B (called the billiard ball) that is held within the boundary of the billiard table. The rules of mathematical billiards are equally simple. The ball B moves within the boundary of the table M with a constant speed. When B hits a boundary of M, the ball ricochets off that side of M according to the law of reflection, namely, the angle of incidence equals the angle of reflection. If the ball reaches a pocket, it stops moving. Otherwise, the ball continues moving indefinitely. Mathematical billiards has many unsolved problems that require a thorough understanding of dynamical systems, complex analysis (in particular Teichmueller Theory) and algebraic geometry. Among these unsolved problems is one proposed by Hubert and Troubetzkoy: Given a set $\mathscr{L}(n)$ which contains all words of length n generated by the encoding of a billiard ball's motion on a polygonal table, what is the growth rate of $\mathscr{L}(n)$ as n approaches infinity? This research is related to a particular sub-problem of Hubert and Troubetzkoy: what happens to the cutting sequences of lines along eigenvector trajectories on the square grid (hence known as eigenlines) when a linear transformation corresponding to the eigenvector is applied to the square grid?

Using a Markov partition of the square grid and identifying identifying it isometrically to the flat torus, an alternative representation of the flat torus that will give a representation of the cutting sequence of an eigenline that uses repeated finite subwords is proposed. Such a partition of the square grid, and thus the flat torus, known as *bi-partition*, will be described in greater detail in this paper. The use of the bi-partition in this paper is drawn mainly from the work of Siemaszko and Wojtkowski, but the conceptual background on Markov partitions in the context of the flat torus is drawn from an article by Bollt and Skufca. The goal of providing a method of

representing the cutting sequence of an irrational line is from Series's article on line foliations of the torus, specifically how irrational lines have dense foliations on the torus. Research in this problem will facilitate for the study of aperiodic cutting sequences of lines with quadratic irrational slopes. Applications of this research outside the field of mathematics include cryptography and the study of dynamical systems emerging from physics and engineering.

Cutting sequences provide a symbolic description of the billiard ball's dynamics and will be the main focus of this research project. A cutting sequence for a ball's path involves reflecting the table over the side that is hit by the ball, turning the path of the ball into a straight line while tessellating the table. In the case of the square billiard table, infinitely repeating this process of tessellation on the real plane creates a square tiling of the plane with lattice points marked, known as a square grid. One can then treat the ball's path as a line crossing the sides of these squares. Labelling parallel sides on the square grid with letters of an alphabet allows one to generate a cutting sequence, which is formed by recording the letters labelling the sides intersected by the line.



The gluing of a square on the square grid.
The tessellation of squares on the square grid. Notice that adjacent sides share the same arrow, hence the isometry between the square grid and the flat torus.

Figure 1

Cutting sequences can also be treated as the encoding of closed, non self-intersecting loops on the surface of a torus. Given the square grid, one can represent the flat torus isometrically by a square, as shown in Figure 1. Gluing, in more mathematical terms, refers to an equivalence relation between two parallel sides of a square, and arrows are typically used to denote the orientation in which the parallel sides are glued. It should be noted that on the square grid, the left side of one square is always the right side of the adjacent square, and likewise with a square's top and bottom sides. Thus, this equivalence relation can be applied to squares on the square grid, generating the flat torus from the square grid by "layering" the individual squares of the square grid on top of one another. This also affects any lines on the square grid by condensing the line onto the flat torus as a series of parallel line segments. In the case of lines with irrational slopes, this "foliation" of the line on the flat torus is dense, meaning it covers every point of the flat torus. On the torus, this property would be equivalent to a loop getting

arbitrarily close to every point on the surface of the torus (this is known as the Kronecker foliation).

The Product is designed to determine the cutting sequence of an eigenline along the stable eigendirection, both directly and by a series of approximations. The Product is composed of two major programs, *cuttingSequence.py* and *eigenvectors.py*, both of which are incorporated into the program *linearTransformationSequence.py*. These components of the Product are all written in Python using the software TextWrangler.

The program *cuttingSequence.py* determines the cutting sequence of a line of the form $y = vx$ given the slope v as an input for a finite number of terms n. The cutting sequence is determined by iteratively comparing the slope of the given line v to vectors with integer components (hence referred to as a comparison integer vector), beginning with the vector <1,1>. If the slope v is less than that of a comparison integer vector $\langle h,k \rangle$, it is assumed that the line intersects a vertical side of the square grid, and this is recorded in the growing cutting sequence. A comparison is then made to the integer vector $\langle h+1, k \rangle$, which would have a slightly lower slope than $\langle h,k \rangle$. If the slope v is greater than the slope of this vector, then it must intersect a horizontal side of the square grid, which is recorded in the cutting sequence. This iterative process of comparisons is repeated n times or until the slope v equals that of its comparison integer vector, in which case the iterative process stops immediately to avoid ambiguity. If the slope v equals the slope of the comparison integer vector, it is known that v is rational, and therefore the sequence generated is periodic, repeating indefinitely. Note that v need not be irrational for this part of the Product.

The second part of the Product, *eigenvectors.py*, randomly generates a 2x2 hyperbolic matrix (the program verifies that the matrix has the properties of a hyperbolic matrix) and determines its expanding eigenvector. After the slope of the expanding eigenvector is recorded, it is input into the function *cuttingSeq()* of *cuttingSequence.py* to directly generate the cutting sequence of the eigenline. In addition to the direction determination of the cutting sequence of the eigenline, the cutting sequence of an integer vector is determined for each application of the hyperbolic matrix to the square grid. The cutting sequence of the vector <1,1> is determined, which is an empty string because a line with a slope of 1 does not intersect either the horizontal or vertical sides of a square. The vector <1,1> is then transformed by the hyperbolic matrix, and a finite number of terms of the cutting sequence of the resulting vector is determined and recorded. This vector is once again transformed by the same matrix, the cutting sequence of the resulting vector is recorded, and the process repeats for a set number of iterations. The slopes and cutting sequences of the iterated integer vectors are compiled in an order list, and from this a trend in the cutting sequences of vectors after an infinite number of applications of the hyperbolic matrix can be empirically determined. The code for the three programs is included in Appendices A-C at the end of this paper. Text in red denotes comments that explain the purpose of and mathematical connection to a part of the program. A sample output of *linearTransformationSequence.py* is included in Appendix D.

It should be noted that a program for determining the continued fraction of a number, *contFrac.py*, was also developed as part of the Product. However, due to floating point imprecision, the accuracy of the program's results was limited and erratic. For this reason, *contFrac.py* was not included in the Final Product. Future research may involve a more accurate

and efficient implementation of the continued fraction algorithm in determining approximations of the cutting sequence of an eigenline.

Continued fractions are a method of representing real numbers using positive integers. Continued fractions provide an alternative to decimal expansions of real numbers. Given a real number *x*, the continued fraction of *x* is,

$$x = n_0 + \frac{1}{n_1 + \frac{1}{n_2 + \ldots}},$$

Where n_i is the largest possible positive integer less than *x* that satisfies the approximation for all $i \geq 0$. Often, the continued fraction of *x* is simply written as $x = [n_0, n_1, n_2, ...]$. The continued fraction of x is found as follows. We define n_0 as the largest integer such that $0 < n_0 < x$. Rearranging the terms in this inequality, it is shown that $0 < x - n_0$. We then define n_1 as the greatest integer such that $0 < n_1 < \frac{1}{x - n_0}$. Hammond's notes on continued fraction algorithms show that the continued fraction of any rational number is finite, meaning a rational number can be expressed as the continued fraction $[n_0, n_1, n_2, ..., n_k]$ for a positive integer *k* (14). By contrast, all irrational numbers have continued fractions with an infinite number of terms. Hammond identifies a particular class of irrationals, which he calls quadratic irrationals, whose continued fractions are periodic. In this context, a continued fraction is periodic if "[a]fter a finite number of terms the sequence of integers repeats cyclically" (Hammond 14). Quadratic irrationals are defined as numbers of the form,

$$\frac{a + b\sqrt{m}}{c},$$

Where *a, b, c, m* $\in \mathbb{Z}$, and $m > 0$ is not a perfect square. This theorem will prove particularly useful when deriving continued fractions from the slopes of lines on the square grid. This method is described in detail by Series's "The Geometry of Markoff Numbers".

Hammond also defines convergents of a continued fraction. Convergents are rational approximations of a irrational number x that are determined by computing a finite number of initial terms in the continued fraction of x. More explicitly, given, $x = [n_0, n_1, n_2, ...]$, the r^{th}-convergent of x is the number represented by the continued fraction $[n_0, n_1, n_2, ..., n_r]$. Because it has a finite continued fraction, the r^{th}-convergent of x must be rational. Given that the slope of an eigenline is irrational (which will be proven later in this paper), the convergents of the slope can be used as approximations of the cutting sequence of the eigenline.

Caroline Series uses the concept of cutting sequences for a geometric representation of irrational numbers. Using isometries between the punctured torus and the hyperbolic plane, Series describes a cutting sequence of a geodesic on the hyperbolic plane that is analogous to that of the square grid. While Series's main purpose is to provide a geometric depiction of a class of irrational numbers known as Markoff irrationalities, the part of interest in Series's article is in the connections between linear maps and cutting sequences. Given a line L on square grid and its cutting sequence (in her article, Series uses the letters a and b to encode vertical and horizontal sides of the squares, respectively), the slope of L can be determined using a technique known as derivation. The derivation of a cutting sequence for a line with a slope greater than 1 sets $a' = ab^n$ and $b' = b$, where n— which is called the *value*— is the number of consecutive b's following an a in the sequence. For example, given the cutting sequence "*abbabbabbbbabbabbbabb"*, the derived sequence sets $a' = ab^2$ and $b' = b$, as 2 is the largest exponent that can simplify that string. Thus, this is a derived sequence of value $n = 2$. The cutting sequence re-codes into $(a')^3(b')^2(a')^2b'a'$, where the exponents denote the uninterrupted repetition of *a'* or *b'*. Series states that if the original sequence is a cutting sequence, the derived

sequence must also be a cutting sequence. By repeatedly deriving the resulting sequence and recording the values of each derived sequence, one obtains information regarding the slope of the original line by writing the values as entries in the continued fraction expansion of the line's slope. Series proves that the derivation of a cutting sequence is equivalent to applying a linear map with the transformation matrix $\begin{bmatrix} 1 & 0 \\ n_k & 1 \end{bmatrix}$, where n_k is the value of the derived sequence. The application of each transformation matrix that corresponds to the k^{th}-derived sequence and recording the cutting sequence of the line after each transformation is equivalent to deriving its cutting sequence k times (Series 21-22). Series shows that the shear transformation described by the aforementioned matrix changes the slope of the line to be its reciprocal in the accordingly transformed basis.

Series' discussion of cutting sequences on the square grid is closely related to the concept of Sturmian sequences. Given a sequence of letters (or factors) u, a "subword" of u is defined as an interval of factors within u. The complexity function $p_u(n)$ is then defined by the number of subwords of length n that can be found within u. For example, given a sequence u = *aabaabaab...*, $p_u(2) = 3$, because there are three sequences of length 2 that can be found within u, namely *ab*, *ba*, and *aa*. However, the subword *bb* will not be found within u. Words that appear in Sturmian sequences will appear an infinite number of times. If one can determine all words of length less than or equal to n that appear in a Sturmian sequence, the sequence can be approximated by its subwords of length n or less. Note that the complexity function of a Sturmian sequence limits what subwords can be found in a given Sturmian sequence u. A sequence u is considered Sturmian if the complexity $p_u(n) = n+1$ for all $n \in \mathbb{N}$. This means that

the complexity is nondecreasing, meaning the number of possible subwords increases with increasing length. Thus, the Sturmian sequence is aperiodic (Arnoux 145-146).

Arnoux also defines Sturmian sequences by the sequence u generated by the equation,

$$u_n = \lfloor \alpha(n+1) + p \rfloor - \lfloor \alpha n + p \rfloor - \lfloor \alpha \rfloor,$$

where $\lfloor x \rfloor$ is the floor function, and $n \in \mathbb{N}$. The numbers α and p are assumed to be irrational numbers in the interval [0,1]. The difference that defines the term u_n is either 0 or 1, the characters used to define cutting sequences in Arnoux's paper (these are analogous to Series's *a* and *b*). Because the constants defining this equation are irrational, there is no regularity or periodicity in the sequence, making it equivalent to the cutting sequence of a line with an irrational slope α and y-intercept p. Thus, the equation essentially determines whether a horizontal or vertical side is intersected in a discrete step of the function. If the difference is 1, then the change in y was greater than or equal to one within a change in x of 1, meaning that a horizontal side must have been intersected. Conversely, if the difference is 0, then the change in y was less than 1, and it did not intersect a horizontal side in the discrete change in x of 1. Therefore, the line must have crossed a vertical side in that discrete step (as the change in x was 1). This numerical definition of Sturmian sequences will prove crucial to the study of eigenline cutting sequences.

We define eigenvectors for the purposes of this research. Given a transformation matrix A, the two eigenvectors of A are vectors that do not change their orientation when A is applied to the plane. By definition, the zero vector is not considered to be an eigenvector of any matrix A, as it would then be an eigenvector of *all* matrices (Stecher 179). Conceptually, the eigenvectors of a 2x2 matrix A determine the general "behavior" of a two-dimensional vector under the

repeated application of A by the linearity of the map. The direction of one eigenvector describes the *stable eigendirection* of A, that is, the direction in which the vectors on the plane will condense after an infinite number of applications of A. Likewise, the direction of the other eigenvector describes the *unstable eigendirection* of A, the direction away from which vectors on the plane will generally shift.

Eigenvectors can have real or complex components, but given the restrictions on acceptable transformation matrices for this research project, the eigenvectors studied will have only real components. The magnitudes of each eigenvector are changed by their corresponding eigenvalues. These eigenvalues are determined by the following equation,

$$A\boldsymbol{v} = \lambda\boldsymbol{v},$$

where the left side of the equation denotes the transformation of the vector **v** under the transformation matrix A, and the right side denotes the scalar multiplication of **v** by the eigenvector λ. This equation can be simplified to,

$$(A - \lambda I)\boldsymbol{v} = \boldsymbol{0},$$

where I is the 2×2 identity matrix. Setting $det(A - \lambda I) = 0$, one can solve for λ through this "characteristic equation" (Strang 287). The eigenvectors are then determined from the eigenvalues, as solutions of **v** for each λ. For example, a 2×2 matrix will have two eigenvalues, and thus has two corresponding eigenvectors.

The matrices studied will be of a subset of the special linear group $SL_2(\boldsymbol{Z})$, which is defined as the group of a 2x2 matrices with a determinant of 1 whose elements are in the set of integers **Z**. Additionally, hyperbolic matrices are defined as having a trace whose absolute value is greater than 2. These qualities ensure that a hyperbolic matrix has only real eigenvalues, as

shown by the following. Given a hyperbolic matrix A with integer elements *a, b, c,* and *d,* the eigenvalues λ_1 and λ_2 can be solved using the characteristic equation as shown:

$$det(A-\lambda I) = 0$$

$(a-\lambda)(d-\lambda) - bc = 0.$ We obtain a quadratic equation and solve it for λ using the quadratic formula:

$$\lambda^2 - (a+d)\lambda + (ad-bc) = 0,$$

$$\lambda = \frac{(a+d) \pm \sqrt{(a+d)^2 - 4(ad-bc)}}{2}$$

Because the determinant $det(A) = ad-bc = 1$, and the trace $tr(A) = a+d$,

$$\lambda = \frac{tr(A) \pm \sqrt{(tr(A))^2 - 4}}{2}.$$

Because $|tr(A)| > 2$ for a hyperbolic matrix, the discriminant in the above solution must be positive. Thus, the two eigenvalues λ_1 and λ_2 of a hyperbolic matrix must both be real quadratic irrational numbers. Using the formula for the eigenvalues it is easy to see that the product of the two eigenvalues of A is 1, meaning the two eigenvalues are multiplicative inverses of each other. The eigenvalue of absolute value larger than 1 corresponds to the stable direction of A.

It can be similarly proven that the slope of the eigenline is quadratic irrational. Given a hyperbolic matrix $A \in SL_2(\mathbf{Z})$,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

it can be shown that the slope of any eigenline of A is irrational.

First, it should be noted that b and c must be nonzero elements of A. By assuming either element equals 0, the determinant $det(A)$ equals to following,

$$det(A) = 1,$$

$$ad - bc = 1,$$

$$ad = 1.$$

Note that because $a,d \in \mathbb{Z}$, $a = d = \pm 1$. However, this means that $|tr(A)| = 2$. A hyperbolic matrix must have a trace whose absolute value is greater than 2, meaning A is no longer hyperbolic! This resulting contradiction proves that $b, c \neq 0$.

Let $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ be an eigenvector of A corresponding to an eigenvalue λ. Using the characteristic equation for determining the eigenvalues of A,

$$\begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \vec{0},$$

A system of equations can be derived.

$$(a-\lambda)v_1 + bv_2 = 0,$$

$$cv_1 + (d-\lambda)v_2 = 0.$$

If one assumes that $v_1 = 0$, the system of equations simplifies to

$$bv_2 = 0$$

$$(d-\lambda)v_2 = 0.$$

By the first equation, $v_2 = 0$, as it has already been shown that $b \neq 0$, showing that $\vec{v} = \vec{0}$. However, by definition, the zero vector cannot be an eigenvector. Thus, we can assume that $\vec{v} \neq \vec{0}$.

Consider, then, the vector $\vec{v} = \begin{bmatrix} 1 \\ v \end{bmatrix}$ to the same eigenvalue λ, giving $\vec{v}$ a slope of v. One can derive a system of equations from the characteristic equation, as above,

$$(a-\lambda) + bv = 0$$

$$c + (d-\lambda)v = 0.$$

By solving for *v* in the first equation, one sees that $v = \frac{\lambda - a}{b}$. Because λ is irrational, and *a* and *b* are both integers, *v* must also be irrational. This means that the cutting sequence for the eigenline is aperiodic, as shown by Arnoux. Explicitly determining the cutting sequence for the eigenline is therefore impractical. However, the eigenline's cutting sequence can be represented using a string of concatenated subwords which are derived using special partitions of the flat torus, as described below.

A *Markov partition* is a method to obtain a symbolic coding of a dynamical system. A Markov partition is first defined by Bollt and Skufca for one-dimensional systems. Given an interval *I* and a map $\tau : I \to I$, define a partition P of I given by points $a_0 < a_1 < a_2 ... < a_n$ in *I*. For $i = 0, ... n-1$, let $I_i = (a_n, a_{n+1})$. If I_i is mapped onto some union of intervals of *P*, then τ is considered to be Markov (1). The authors then generalize this definition for higher-dimensional systems, stating that given a metric space *M*, a certain partition *P* of *M*, and the map $f : M \to M$, the map *f* has a corresponding Markov partition if the image of the map of an element of *P* stretches across the preimage in the expanding direction and is enclosed by the preimage in the contracting direction (2). In the context of this research, the image of the map refers to a partition of the square lattice along the expanding and contracting eigendirections of a given hyperbolic matrix, known as a *bi-partition*.

Given that the square lattice on the real plane is isometric to the flat torus $\mathbb{T}^2$, a Markov partition of the flat torus can be formed based on the stable and unstable eigendirections of a hyperbolic matrix *A*. Siemaszko and Wojtkowski describe a *bi-partition* of the torus as follows: consider two rectangles R_1 and R_2 with horizontal side lengths *u, v,* and vertical side lengths, *p, q,* respectively. The rectangles are adjacent to one another such that the vertical side of R_1 is against

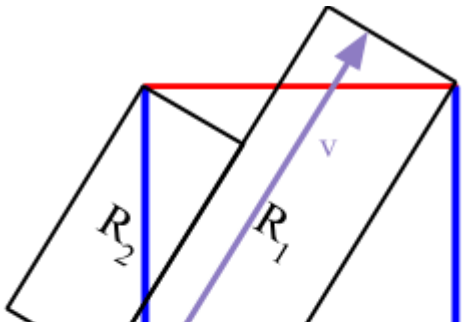
that of R_2. A "lattice of translations *L*" is generated the vectors *[v, p]* and *[-u, q]*, translating the partition $\{R_1, R_2\}$ across the plane, tiling it in a manner similar to the square grid. This partition is known as the bi-partition (2384-2385). The generator for the tiling by translates of the bi-partition has the origin as a fixed point on its boundary.

For the purposes of this paper, the bi-partition is "tilted" so that the horizontal sides lie along the stable eigendirection, and the vertical sides the unstable eigendirection, that is, each side of a rectangle is parallel either to the stable eigendirection or the unstable eigendirection.

This "tilting" action is really the use of the expanding and contracting eigenvectors of *A*— henceforth $\vec{v_1}$ and $\vec{v_2}$, respectively— as the basis vectors in place of the standard basis. This change of basis allows the hyperbolic matrix *A* in the standard basis to be represented as the diagonal matrix *B* in the basis $(\vec{v_1}, \vec{v_2})$,

$$B = \begin{bmatrix} \mu & 0 \\ 0 & \frac{1}{\mu} \end{bmatrix},$$

Where μ is the expanding eigenvalue of *A* (recall that the eigenvalues of a hyperbolic matrix are multiplicative inverses of each other). This diagonalization of the hyperbolic matrix is drawn from Stecher's explanation of a change of basis (182). That the nonzero elements of *B* are inverses of one another is implied in Siemaszko and Wojtkowski's description of a "hyperbolic toral automorphism" using a basis of the stable and unstable directions of a system. The above representation of *B* is simply a special case of Siemaszko and Wojtkowski's definition of a hyperbolic toral automorphism for eigenlines oriented along the stable direction (that is, the stable eigendirection). Using this new basis facilitates the transformation of the bi-partition under

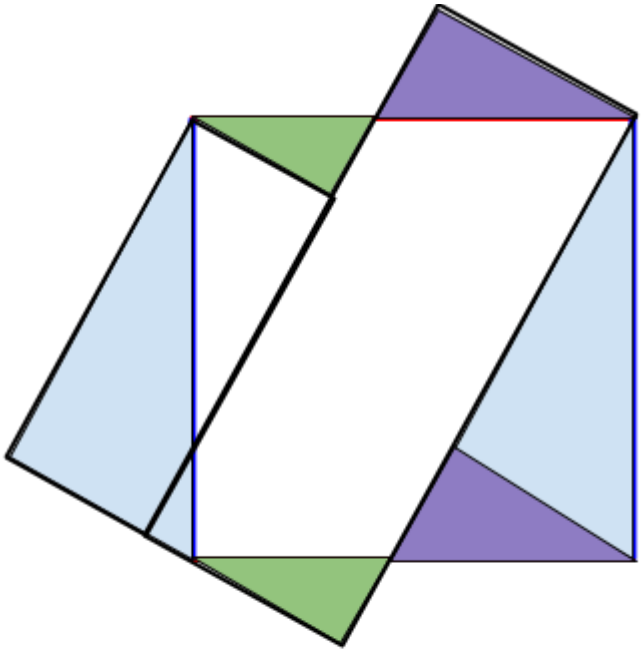


the hyperbolic matrix. Rather than having to map the bi-partition using A in the standard basis, the same points in the bi-partition are mapped using B in the basis of the eigenvectors of A. The lattice of translations that describe the tiling of the bi-partition on the plane can then be made using the basis $(\vec{v_1}, \vec{v_2})$. The transformation of the bi-partition under B in the basis $(\vec{v_1}, \vec{v_2})$ is simply an expansion of the horizontal side lengths of R_1 and R_2 by a factor of μ, and a contraction of the vertical side lengths by a factor of μ^{-1}.

It is important to note that the bi-partition must only have integer points of the square grid on the boundary of the bi-partition. This stipulation is based on the isometry between the flat torus and the square grid, where all integer points on the square grid are mapped to a single point. Assume that there lies an integer point *(a,b)* inside a rectangle of the generating bi-partition (whose boundary contains the origin). All integer points in the plane are mapped to the origin of the torus. In particular, the point *(a,b)* is mapped to the origin. However, this means that the interior of the bi-partition intersects its own boundary. The bi-partition and square grid are both isometric to the flat torus and thus one another, and because the integer points on the flat torus are mapped to the origin alone, the self-intersection of the bi-partition would result in a contradiction. Thus, there cannot be any integer points of the square grid within the boundary of the bi-partition.

With this restriction on the boundary of the bi-partition defined, the dimensions of the rectangles R_1 and R_2 can be uniquely defined for a fixed or determined basis of eigenvectors of a hyperbolic matrix using the integer points at a vertex of each rectangle, as each side of either rectangle can contain at most one integer point. Because neither rectangle can enclose an integer point, the integer points on a unit square of the square grid in the standard basis serve as bounds

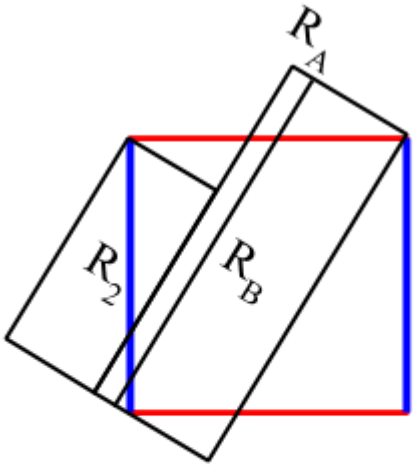
for the dimensions of R_1 and R_2. It is henceforth assumed that the length and width of R_2 are smaller than the length and width of R_1.



Parts of the bi-partition "outside" of the square defining the flat torus can simply be mapped to the interior.

Figure 3

It can be shown graphically by cutting and translating triangles composing the bi-partition that there exists a bijection between the bi-partition and the flat torus, as evidenced by their equal areas in Figure 3. Using the equivalence relation that defines the flat torus on the square grid, sections of the bi-partition that are "outside" the boundary of the flat torus are moved correctly to the interior, defining the same points on the torus. Thus, through these cuts and translations we see that the bi-partition creates an isometry with the flat torus.



R_1 of the bi-partition is subdivided into R_B and R_A.
R_B has a width equal to that of R_2.

Figure 4

By tiling the plane by translations of the bi-partition, one can label sides of the bi-partition and generate a cutting sequence for lines that cross these labelled sides, as in the case of the flat torus. However, sides that are alongside each other due to adjacent translations of the bi-partition are not necessarily the same length. To facilitate the labelling of the boundary of bi-partition, the larger rectangle, R_1, of each translation of the bi-partition is further subdivided along a particular expanding eigenline L. The cut forms two rectangles with the same length as R_1, but the width of the rectangle below L must be equal to that of the smaller R_2. The rectangle above the eigenline will be denoted as R_A, and the rectangle below as R_B. This further subdivision of the bi-partition defines an *augmented bi-partition*.

Using the augmented bi-partition, the gluing of its boundary becomes more apparent, as R_2 and R_B now have the same width. Given an augmented bi-partition β defined by the eigenvectors of a hyperbolic matrix A, if an eigenline E of matrix A intersects opposite sides of a

rectangle R_i of β, we state that E "crosses" R_i. By recording the sequence of vertical and horizontal sides intersected by E within the boundary of R_i, one can generate a finite word w_i that corresponds to R_i for all i defining a rectangle of β. The word w_i can be thought of as a sample of the cutting sequence of E taken from within the boundary of the rectangle R_i, recording the sequence of sides intersected by E within R_i. Because the bi-partition β tiles the plane by translations, the word w_i can be associated with any translate of R_i on the real plane, that is, the subword w_i is independent of the translation of the bi-partition by the lattice of translations L.

The cutting sequence of E can therefore be thought of as the concatenation of an infinite sequence of finite words. The order of these finite words in the cutting sequence is determined by the order of corresponding rectangles of β that are crossed by E. Any two eigenlines distinct E_1 and E_2 that cross the same pattern of rectangles in β must then have the same pattern of corresponding finite words in their cutting sequences, and must therefore have the same cutting sequences.

Recall that the application of the hyperbolic matrix A to the corresponding bi-partition β readjusts the lengths and widths of rectangles R_1 and R_2 that define the bi-partition. More specifically, given the eigenvalue for the expanding eigenvector λ, the lengths of the rectangles are stretched by a factor of λ and the width contracted by a factor of λ^{-1}. Applying the matrix A to β must then change the encoding of the finite word w_i associated with each rectangle R_i of β. The eigenline E will cross the rectangles of β in the same order, as the orientation of the rectangles is not changed upon applying the matrix A. Similarly, applying the matrix A will change the coding of each word w_i for each rectangle R_i, but the *order* of the words in the cutting sequence does not change. Each word w_i is substituted with another finite word w'_i that encodes

the pattern of sides of the square torus intersected by E within each transformed rectangle R_i. The cutting sequence of E after the transformation by A is therefore represented by replacing each word w_k in a sequence with the word w'_k, for $k \in \mathbb{N}$.

This method of obtaining the cutting sequence of E applies for any number of applications of the matrix A to the square grid, substituting each finite word in the cutting sequence with a corresponding replacement word. This iterated substitution can determine the cutting sequence of E for any number of applications of A, providing an implicit representation of the cutting sequence using finite words.

Appendix A: cuttingSequence.py

```
'''
Generates a cutting sequence for a line with a given slope.

NOTE: This program assumes that the line passes through the origin, hence why there is no input for the y-intercept.
Such a consideration is not pertinent to the theoretical component of the Paper.

'''

import math

# RECURSIVE PROCESS

# Take vector <1,1> (whose slope is 1) and compare to the given v's slope
#	If v's slope is less than that of <1,1>, then compare to <2,1>; Record vertical side being struck by ball
#	If v's slope is greater than that of <1,1>, then compare to <1,2>; Record horizontal side being struck by ball
# Repeat process with successively better linear approximations

'''

Compares slope of given vector to a set of approximation vectors (with integer components)
and outputs the cutting sequence for a line along the given vector

Encoding:
Horizontal side intersected, "A"
Vertical side intersected, "B"

@param:
	- input_slope (float): slope of input line
	- iter (int): iterations of program remaining (to avoid an endless loop)
	- sequence (string): encodes the cutting sequence of a line with slope = "input_slope". sequence
	is concatenated at each iteration of the function
	- comp_x: the x-component of the comparison integer vector used to determine whether
	the line with "input_slope" intersects a horizontal or vertical line. Increments if vertical side is hit.
	- comp_y:  the y-component of the comparison integer vector used to determine whether
```

```
        the line with "input_slope" intersects a horizontal or vertical line. Increments if
horizontal side is hit.

    '''

    #=== Function Definitions ===#

    def cuttingSeq(input_slope, iter, sequence, comp_x = 1, comp_y = 1):

        if iter <= 0:
            return sequence

        comp_slope = (comp_y * 1.0)/ comp_x
#       print comp_slope

        if input_slope > comp_slope:
#           print "A"
            sequence = sequence + "A"
            return cuttingSeq(input_slope, iter - 1, sequence, comp_x, comp_y + 1);

        elif input_slope < comp_slope:
#           print "B"
            sequence = sequence + "B"
            return cuttingSeq(input_slope, iter - 1, sequence, comp_x + 1, comp_y);
        else:
            return sequence
```

Appendix B: *eigenvectors.py*

```
'''
eigenvectors.py

Takes a given matrix, checks that it is hyperbolic, and returns the expanding eigenvector and its slope

'''

import numpy as np
import math
from random import *

#=== Function Definitions ===#

'''
tests if a given matrix has a determinant of 1
'''
def detTest(matrix):
	print round(np.linalg.det(matrix)) == 1

'''
tests if a given matrix is hyperbolic (i.e. has determinant of 1 and a trace whose absolute value is greater than 2)
'''
def isHyperbolic(matrix):
	if np.trace(matrix) > 2 and round(np.linalg.det(matrix)) == 1:
		return True;
	return False;

'''

@param:
- matrix: 2-by-2 matrix with real entries

@return: returns the eigenvector of the given hyperbolic matrix whose slope is positive
'''
def eigenHyp(matrix):

	list = []

	if isHyperbolic(matrix):
		w, v = np.linalg.eig(matrix)
```

```
                largestEigenvalue = 0
                termOfLargest = -1

                for i in range(len(v)):
                        if abs(w[i]) > abs(largestEigenvalue):
                                largestEigenvalue = w[i]
                                termOfLargest = i

                return v[termOfLargest]
```

```
        '''
        Determines slope of a 2-by-1 vector (which can encode the slope of a line crossing through it)

        @return: returns slope of given vector
        '''
        def slope(vector):
                return (1.0*vector.item(1))/vector.item(0);


        '''
        Randomly generates a hyperbolic matrix, checking that the matrix formed is hyperbolic

        @return: returns a random hyperbolic matrix with elements of value "> 0" and "< max"
        '''
        def genMatrix(max):

                # initialize
                a = randint(0,max)
                b = 0
                c = 0
                d = randint(0,max)

                while abs(a+d) <= 2:
                        d = randint(0,max)


                bc = (a*d) - 1

                if a == 0 or d == 0:
                        b = 1
                        c = -1
                else:
                        if isPrime(bc):
                                b = bc
```

```
                    c = 1
                else:
                    b = randFactor(bc)
                    c = bc/b
            retMatrix = np.matrix([[a, b], [c, d]])

            if isHyperbolic(retMatrix):
                return retMatrix

        '''
        Checks if an integer is prime. Speeds up calculations in determining 'b' and 'c' in the
genMatrix() function

        @param:
        - num (integer): an integer input

        @return: a boolean stating whether "num" is prime
        '''
        def isPrime(num):

            if num < 2:
                return False;

            for i in range(2,num):
                if num % i == 0:
                    return False;

            return True;

        '''
        Returns a random factor of a given integer.
        Speeds up calculations in determining 'b' and 'c' in the genMatrix() function, should b*c
not be prime

        @param:
        - num (integer): an integer input

        @return: a random factor of "num"
        '''
        def randFactor(num):
            if isPrime(num):
                return 1

            list = []
```

```
for i in range(2,num):
        if num % i == 0:
                list.append(i)

return list[randint(0,len(list) - 1)]
```

Appendix C: *linearTransformationSequence.py*

```
'''

linearTransformSequence.py

Incorporates eigenvectors.py and cuttingSequence.py to generate the cutting sequences of the
expanding eigenline and an approximation vector that is repeatedly mapped by a hyperbolic
matrix.

'''

#=== Import Statements ===#

import math
import numpy as np

import cuttingSequence as cutSeq
import eigenvectors as eig

#=== Constants ===#

WORD_LENGTH_MAX = 50 # Maximum number of terms in cutting sequence
MATRIX_APPLY_MAX = 15 # Number of times transformation matrix is applied to testVector
MATRIX_ELEMENT_MAX = 9 # max value of elements in the randomly-generated hyperbolic
# matrix

#=== Variables ===#

matrix = eig.genMatrix(MATRIX_ELEMENT_MAX) # randomly generated hyperbolic matrix

expandingEigen = eig.eigenHyp(matrix) # expanding eigenvector of matrix
slope = eig.slope(expandingEigen) # slope of expanding eigenvector

testVector = np.matrix('1; 1') # Initial comparison integer vector

seq = "" # string used for storing the cutting sequence of a given line

#=== Function Definitions ===#
'''
Generates a list of approximation vectors by iterating the application of the hyperbolic matrix
onto an integer vector <1,1>

@param:
- initVector: the initial integer vector being tested
```

```
- matrix: the hyperbolic matrix iteratively applied to "initVector" MATRIX_APPLY_MAX
times,
recording each resulting vector as an iteration

@return: generates a list of approximation vectors
'''
def approxVectors(initVector, matrix):

        returnList = []
        iterVector = np.matrix([[initVector.item(0)], [initVector.item(1)]])

        for i in range(0,MATRIX_APPLY_MAX):
                returnList.append(iterVector)
                iterVector = matrix.dot(iterVector)

        return returnList

'''
Generates a list of the slopes of the approximation vectors that corresponds to the list of
approximation vectors

@param:
- approxVectorList: list of vectors resulting from the repeated application of "matrix" onto a
"testVector"

@return: list of the slopes of the approximation vectors
'''
def approxSlopes(approxVectorList):
        returnList = []
        for i in approxVectorList:
                returnList.append(eig.slope(i))
        return returnList

'''
Generates a list of the cutting sequences of the approximation vectors
that corresponds to the list of the slopes of the approximation vectors

@param:
- approxSlopeList: list of slopes of vectors resulting from the repeated application of "matrix"
onto a "testVector"

@return: list of the cutting sequences of the approximation vectors
'''
def approxSequences(approxSlopeList):
        returnList = []
```

```
        for i in approxSlopeList:
                seq = ""
                returnList.append(cutSeq.cuttingSeq(i, WORD_LENGTH_MAX, seq, 1, 1))
        return returnList

#=== Function Executions ===#

approxVectorList = approxVectors(testVector,matrix)
approxSlopeList = approxSlopes(approxVectorList)

# print approxVectorList
print "Hyperbolic Matrix: " + str(matrix)
print "\n" # line break for clarity in output

print "Eigenvector Tested: {0}".format(expandingEigen)
print "\n"

print "Eigenvector Slope: {0}".format(slope)
print "\n"

print "Slope list: \n {0}".format(approxSlopeList)
print "\n"

print "Sequence List: \n {0}".format(approxSequences(approxSlopeList))

'''
Direct determination of cutting sequence of eigenline.
Note that that absolute value of the slope is taken so that, should "slope" be negative, the cutting
sequence can still be generated.
Taking the absolute value of the slope does not affect its cutting sequence because the square
grid is reflected along the x-axis.
Therefore, the cutting sequence is identical between two slopes that are negatives of one antoher.
'''
print "Expanding Eigenvector Cutting Sequence: \n"
print cutSeq.cuttingSeq(1.0*abs(slope), WORD_LENGTH_MAX, seq, 1, 1)
```

Appendix D: Sample Output

Hyperbolic Matrix: [[7 13], [1 2]]

Expanding Eigenvector Tested: [[0.98962357, -0.8836437]]

Expanding Eigenvector Slope: -0.892908914639

Slope list for approximation vector after repeated applications of hyperbolic matrix:
[1.0, 0.15, 0.1452513966480447, 0.1451917033312382, 0.1451909476661952, 0.1451909380992926, 0.14519093797817315, 0.14519093797663973, 0.14519093797662033, 0.14519093797662008, 0.14519093797662008, 0.14519093797662008, 0.14519093797662008, 0.14519093797662008, 0.14519093797662008]

Corresponding Sequence List:
['', 'BBBBBBABBBBBBBABBBBBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB',
'BBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBBBBBBABBB']

Expanding Eigenvector Cutting Sequence:
BABABABABABABABABBABABABABABABABABBABABABABABABABA

Works Cited

Arnoux, Pierre. “Sturmian Sequences.” *Substitutions in Dynamics, Arithmetics and Combinatorics. Lecture Notes in Mathematics*, vol 1794, pp. 143-198, https://link.springer.com/chapter/10.1007%2F3-540-45714-3_6.

Bollt and Skufca. “Markov Partitions.” *Encyclopedia of Nonlinear Science*, 2005. https://webspace.clarkson.edu/~ebollt/Papers/MarkovPartitionsNonlinearEncyclopedia.pdf. Accessed 27 Mar 2018.

Hammond, William. “Continued Fractions and the Euclidean Algorithm.” University of Albany, pp. 2-15, https://www.math.u-bordeaux.fr/~pjaming/M1/exposes/MA2.pdf.

Series, Caroline. “The Geometry of Markoff Numbers.” *The Mathematical Intelligencer*, vol. 7, no. 3, 1985, pp. 20-29, https://perso.univ-rennes1.fr/serge.cantat/Documents/series-geometry-markoff.pdf.

Siemaszko A, and Maciej Wojtkowski. “Counting Berg Partitions.” *Nonlinearity*, vol. 24, 2011, pp. 2383-2393, https://www.researchgate.net/publication/45930802_Counting_Berg_partitions. Accessed 25 Mar 2018.

Stecher, Michael. Linear Algebra. College Station: Texas A&M UP. Web. 27 Mar 2018. http://www.math.tamu.edu/~stecher/304/text.shtml.

Strang, Gilbert. “Introduction to Linear Algebra, Fifth Edition (2016).” MIT Math, pp. 283-297, http://math.mit.edu/~gs/linearalgebra/ila0601.pdf.